More Bang for Your Bug

Slides At
http://mitigation.games

syscall(sys_geteuid)

Tomorrow: Birds of a Feather

Linux CVEs Open Discussion

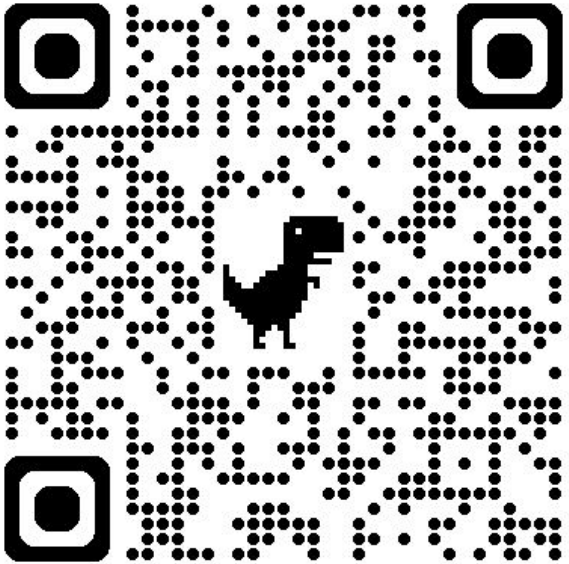
Sep 20, 2024, 12:45 PM

45m

"Room 1.85 - 1.86" (Austria Center)

Speaker

Damiano Melotti (Google)



Why Triage CVEs in 2024

Why Triage CVEs in 2024

What are vulns anyway?

The patch

https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=0ebc1064e4874d5987722a2ddbc18f94aa53b211

netfilter: nf_tables: disallow rule addition to bound chain via NFTA_RULE_CHAIN_ID

```
Bail out with EOPNOTSUPP when adding rule to bound chain via
NFTA_RULE_CHAIN_ID. The following warning splat is shown when
adding a rule to a deleted bound chain:

 WARNING: CPU: 2 PID: 13692 at net/netfilter/nf_tables_api.c:2013 nf_tables_chain_destroy+0x1f7/0x210 [nf_tables]
 CPU: 2 PID: 13692 Comm: chain-bound-rul Not tainted 6.1.39 #1
 RIP: 0010:nf_tables_chain_destroy+0x1f7/0x210 [nf_tables]

Fixes: d0e2c7de92c7 ("netfilter: nf_tables: add NFT_CHAIN_BINDING")
Reported-by: Kevin Rich <kevinrich1337@gmail.com>
Signed-off-by: Pablo Neira Ayuso <pablo@netfilter.org>
Signed-off-by: Florian Westphal <fw@strlen.de>
```

The patch

https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=0ebc1064e4874d5987722a2ddbc18f94aa53b211

netfilter: nf_tables: disallow rule addition to bound chain via NFTA_RULE_CHAIN_ID

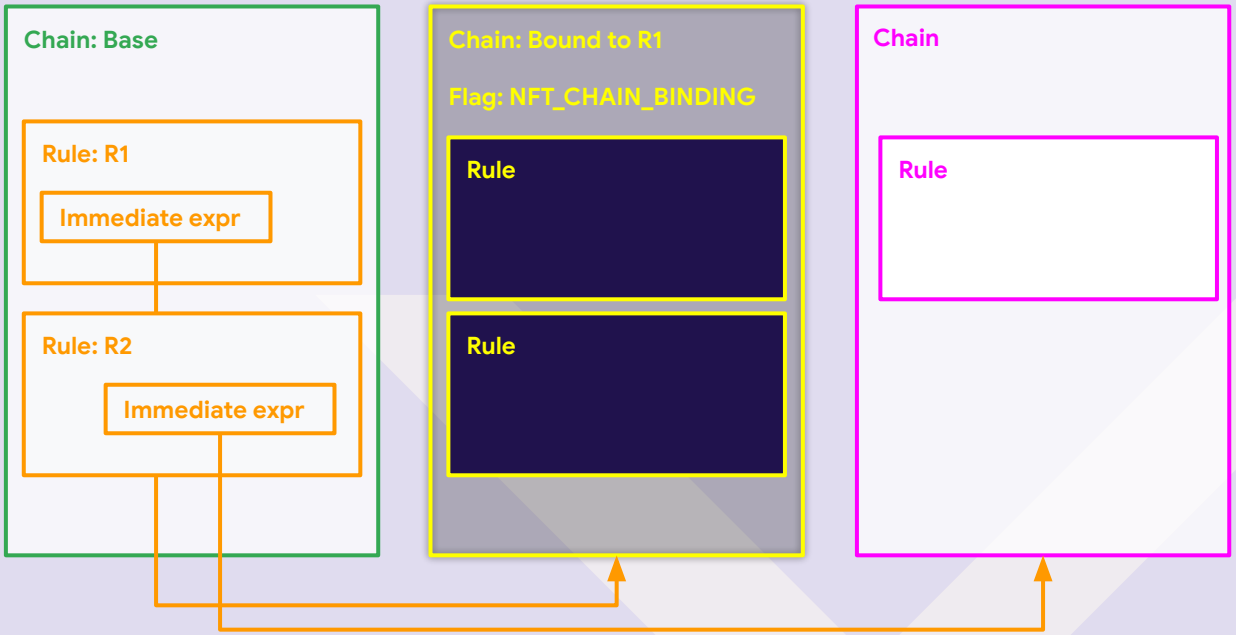
```
Bail out with EOPNOTSUPP when adding rule to bound chain via
NFTA_RULE_CHAIN_ID. The following warning splat is shown when
adding a rule to a deleted bound chain:

 WARNING: CPU: 2 PID: 13692 at net/netfilter/nf_tables_api.c:2013 nf_tables_chain_destroy+0x1f7/0x210 [nf_tables]
 CPU: 2 PID: 13692 Comm: chain-bound-rul Not tainted 6.1.39 #1
 RIP: 0010:nf_tables_chain_destroy+0x1f7/0x210 [nf_tables]

Fixes: d0e2c7de92c7 ("netfilter: nf_tables: add NFT_CHAIN_BINDING")
Reported-by: Kevin Rich <kevinrich1337@gmail.com>
Signed-off-by: Pablo Neira Ayuso <pablo@netfilter.org>
Signed-off-by: Florian Westphal <fw@strlen.de>
```

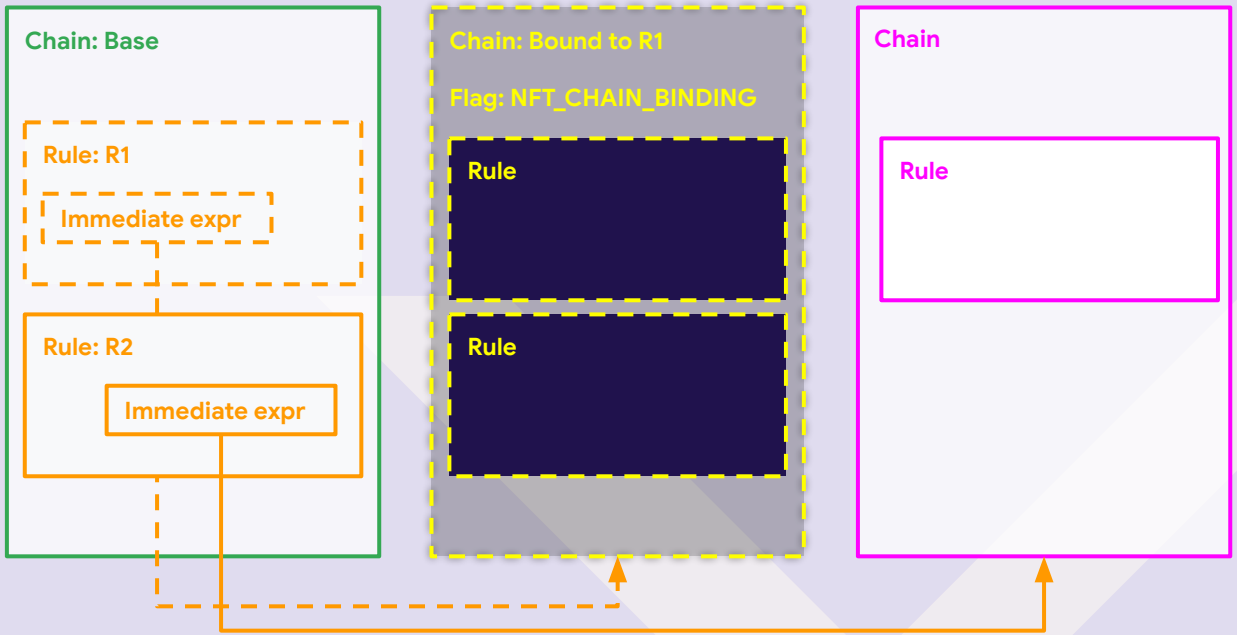
The original code

https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=d0e2c7de92c7

netfilter: nf_tables: add NFT_CHAIN_BINDING

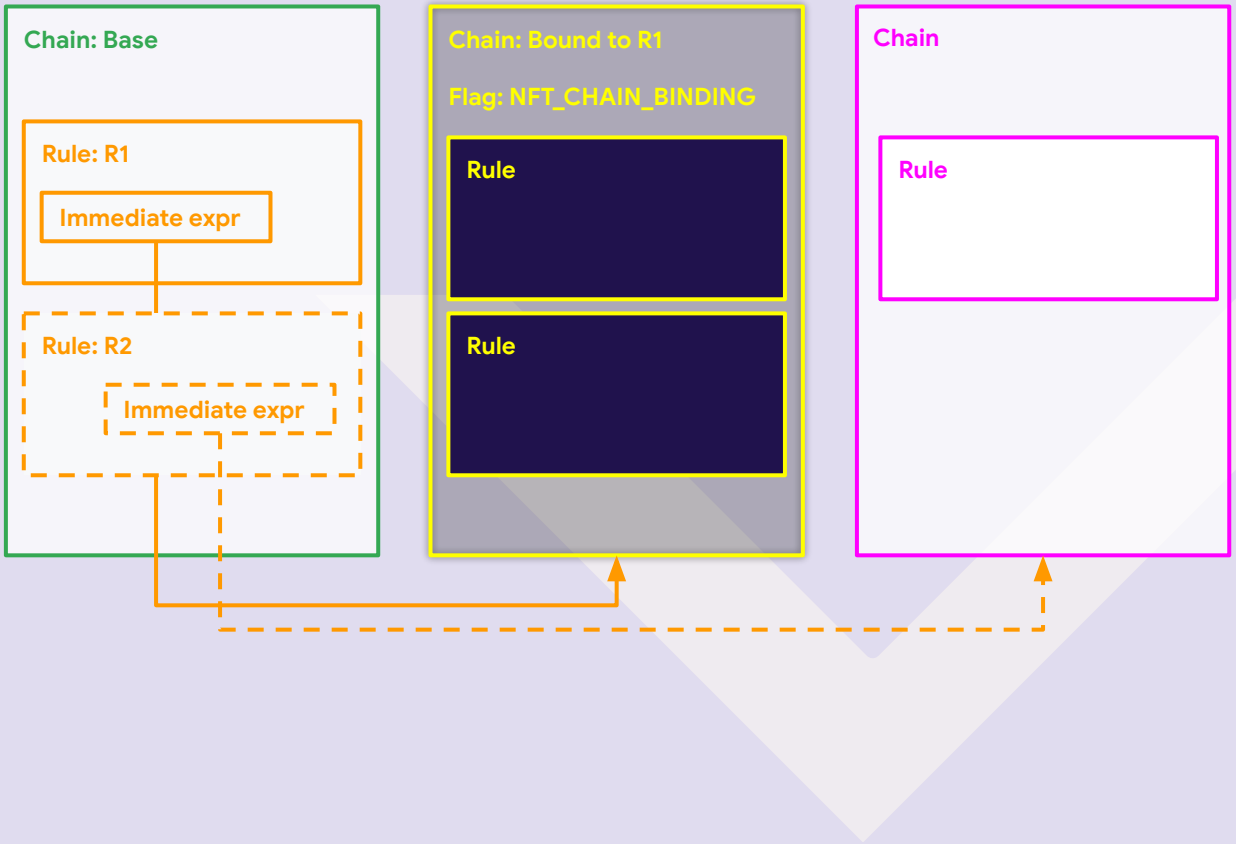
```
This new chain flag specifies that:

* the kernel dynamically allocates the chain name, if no chain name
  is specified.

* If the immediate expression that refers to this chain is removed,
  then this bound chain (and its content) is destroyed.
```


Chain: Base
Rule: R1
Immediate expr
Rule: R2
Immediate expr
Chain: Bound to R1
Flag: NFT_CHAIN_BINDING
Rule
Rule
Chain
Rule



NFTs

* If the immediate expression that refers to this chain is removed, then this bound chain (and its content) is destroyed.


Chain: Base
Rule: R1
Immediate expr
Rule: R2
Immediate expr
Chain: Bound to R1
Flag: NFT_CHAIN_BINDING
Rule
Rule
Chain
Rule
NFTs

The patch

https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=0ebc1064e4874d5987722a2ddbc18f94aa53b211

netfilter: nf_tables: disallow rule addition to bound chain via NFTA_RULE_CHAIN_ID

```
Bail out with EOPNOTSUPP when adding rule to bound chain via
NFTA_RULE_CHAIN_ID. The following warning splat is shown when
adding a rule to a deleted bound chain:

 WARNING: CPU: 2 PID: 13692 at net/netfilter/nf_tables_api.c:2013 nf_tables_chain_destroy+0x1f7/0x210 [nf_tables]
 CPU: 2 PID: 13692 Comm: chain-bound-rul Not tainted 6.1.39 #1
 RIP: 0010:nf_tables_chain_destroy+0x1f7/0x210 [nf_tables]
```

The patch

https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=0ebc1064e4874d5987722a2ddbc18f94aa53b211

Diffstat

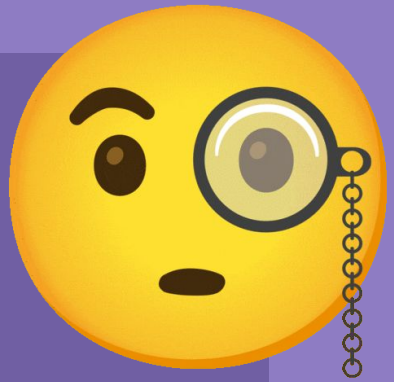
-rw-r--r-- net/netfilter/nf_tables_api.c 5

1 files changed, 3 insertions, 2 deletions

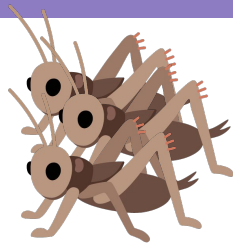
```
diff --git a/net/netfilter/nf_tables_api.c b/net/netfilter/r
index b9a4d3fd1d3487..d3c6ecd1f5a680 100644
--- a/net/netfilter/nf_tables_api.c
+++ b/net/netfilter/nf_tables_api.c
@@ -3811,8 +3811,6 @@ static int nf_tables_newrule(struct sk
                        NL_SET_BAD_ATTR(extack, nla[NFTA_RUL
                        return PTR_ERR(chain);
                }
-               if (nft_chain_is_bound(chain))
-                       return -EOPNOTSUPP;

        } else if (nla[NFTA_RULE_CHAIN_ID]) {
                chain = nft_chain_lookup_byid(net, table, nl
@@ -3825,6 +3823,9 @@ static int nf_tables_newrule(struct sk
                return -EINVAL;
        }

+       if (nft_chain_is_bound(chain))
+               return -EOPNOTSUPP;
+
        if (nla[NFTA_RULE_HANDLE]) {
                handle = be64_to_cpu(nla_get_be64(nla[NFTA_R
                rule = __nft_rule_lookup(chain, handle);
```

```
26
27      if (nla[NFTA_RULE_CHAIN]) {
28              chain = nft_chain_lookup(net, table, nla[NFTA_RULE_CHAIN],
29                                       genmask);
30              if (IS_ERR(chain)) {
31                      NL_SET_BAD_ATTR(extack, nla[NFTA_RULE_CHAIN]);
32                      return PTR_ERR(chain);
33              }
34              if (nft_chain_is_bound(chain))
35                      return -EOPNOTSUPP;
36
37      } else if (nla[NFTA_RULE_CHAIN_ID]) {
38              chain = nft_chain_lookup_byid(net, nla[NFTA_RULE_CHAIN_ID]);
39              if (IS_ERR(chain)) {
40                      NL_SET_BAD_ATTR(extack, nla[NFTA_RULE_CHAIN_ID]);
41                      return PTR_ERR(chain);
42              }
43      } else {
44              return -EINVAL;
45      }
46
47      if (nla[NFTA_RULE_HANDLE]) {
48              handle = be64_to_cpu(nla_get_be64(nla[NFTA_RULE_HANDLE]));
49              rule = __nft_rule_lookup(chain, handle);
50              if (IS_ERR(rule)) {
51                      NL_SET_BAD_ATTR(extack, nla[NFTA_RULE_HANDLE]);
52                      return PTR_ERR(rule);
53              }
54
55              if (nlh->nlmsg_flags & NLM_F_EXCL) {
56                      NL_SET_BAD_ATTR(extack, nla[NFTA_RULE_HANDLE]);
57                      return -EEXIST;
58              }
59              if (nlh->nlmsg_flags & NLM_F_REPLACE)
60                      old_rule = rule;
61              else
62                      return -EOPNOTSUPP;
63      } else {
64              if (!(nlh->nlmsg_flags & NLM_F_CREATE) ||
```

Why was this change necessary?



No discussion on lkml



Linux 6.5-rc4
- by Linus Torvalds @ 2023-07-30 20:38 UTC [3%]

[GIT PULL] Networking for 6.5-rc4
- by Paolo Abeni @ 2023-07-27 13:12 UTC [5%]

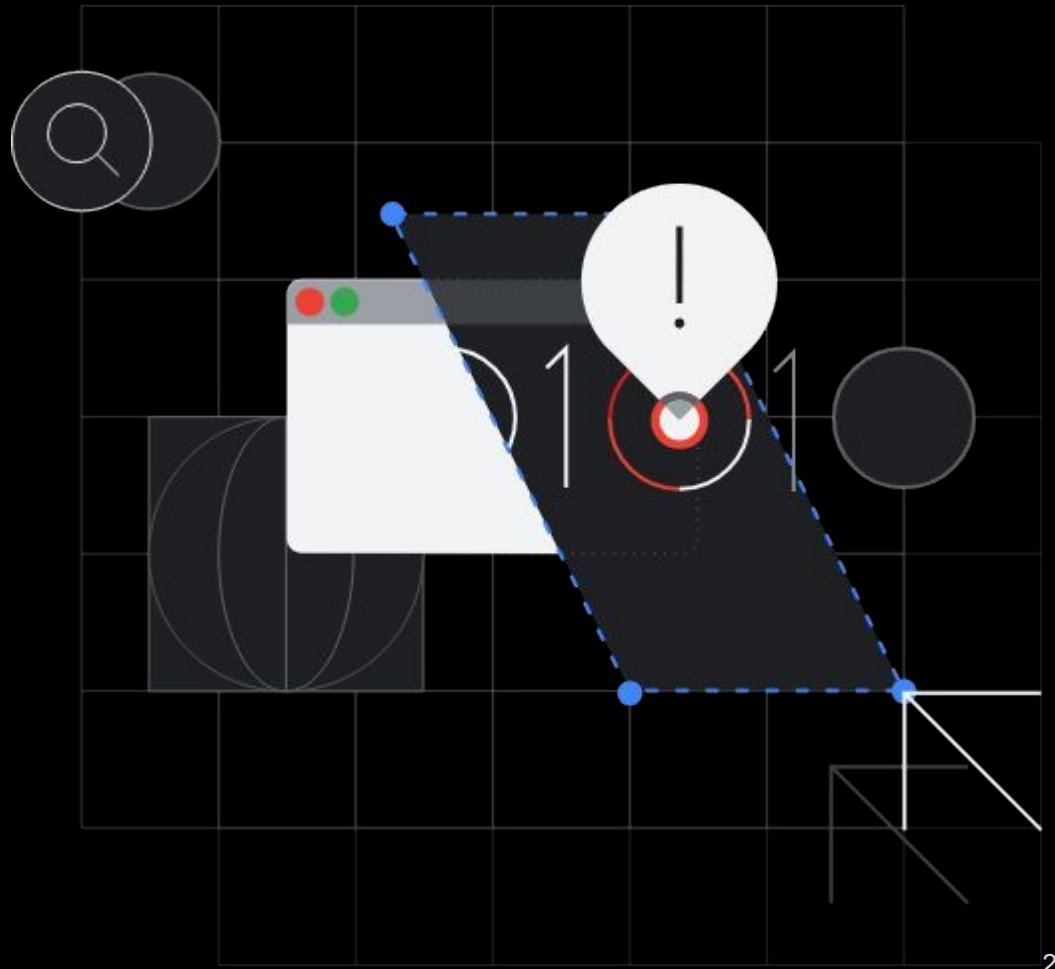
Re: [PATCH net 1/3] netfilter: nft_set_rbtree: fix overlap expiration walk
- by patchwork-bot+netdevbpf @ 2023-07-27 5:20 UTC [7%]

[PATCH net 3/3] netfilter: nf_tables: disallow rule addition to bound chain via NFTA_RULE_CHAIN_ID
- by Florian Westphal @ 2023-07-26 15:23 UTC [25%]

[PATCH net 0/3] netfilter fixes for net
- by Florian Westphal @ 2023-07-26 15:23 UTC [10%]

[PATCH nf] netfilter: nf_tables: disallow rule addition to bound chain via NFTA_RULE_CHAIN_ID
- by Pablo Neira Ayuso @ 2023-07-23 14:41 UTC [26%]

kernelCTF



kernelCTF - How To Submit an Exploit

Steal Flag

kernelCTF - How To Submit an Exploit

Steal Flag

Fix Bug Upstream (for 0day)

kernelCTF - How To Submit an Exploit

Wait 30 Days

kernelCTF - How To Submit an Exploit

Wait 30 Days

Submit Exploit & Writeup

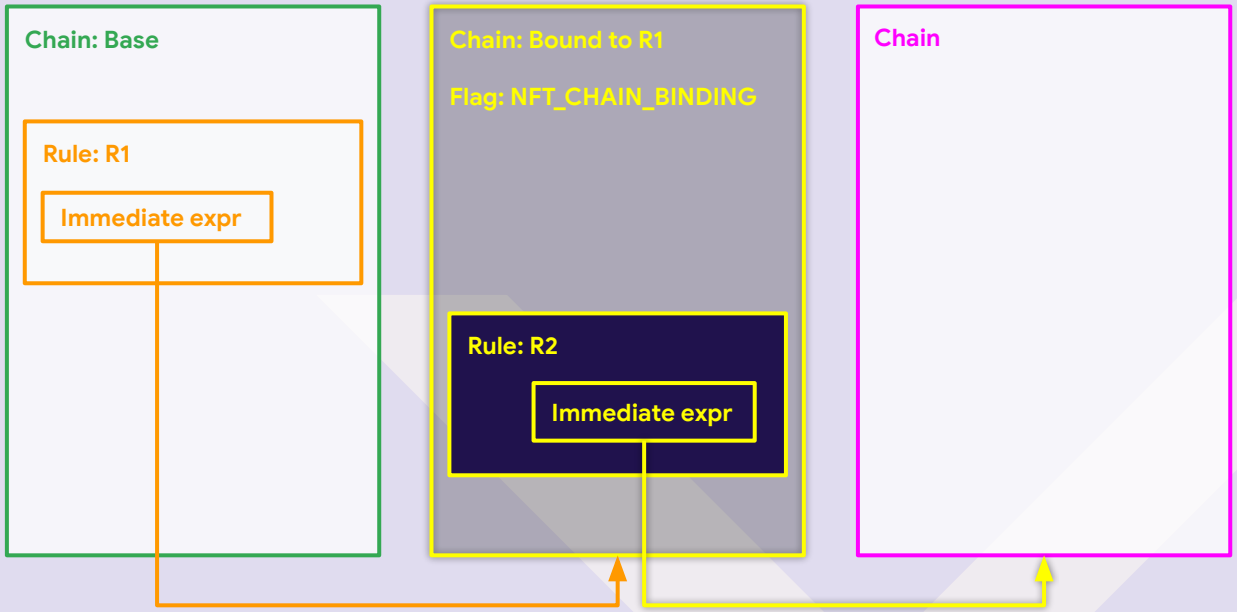
kernelCTF - How To Submit an Exploit

Collect Bounty

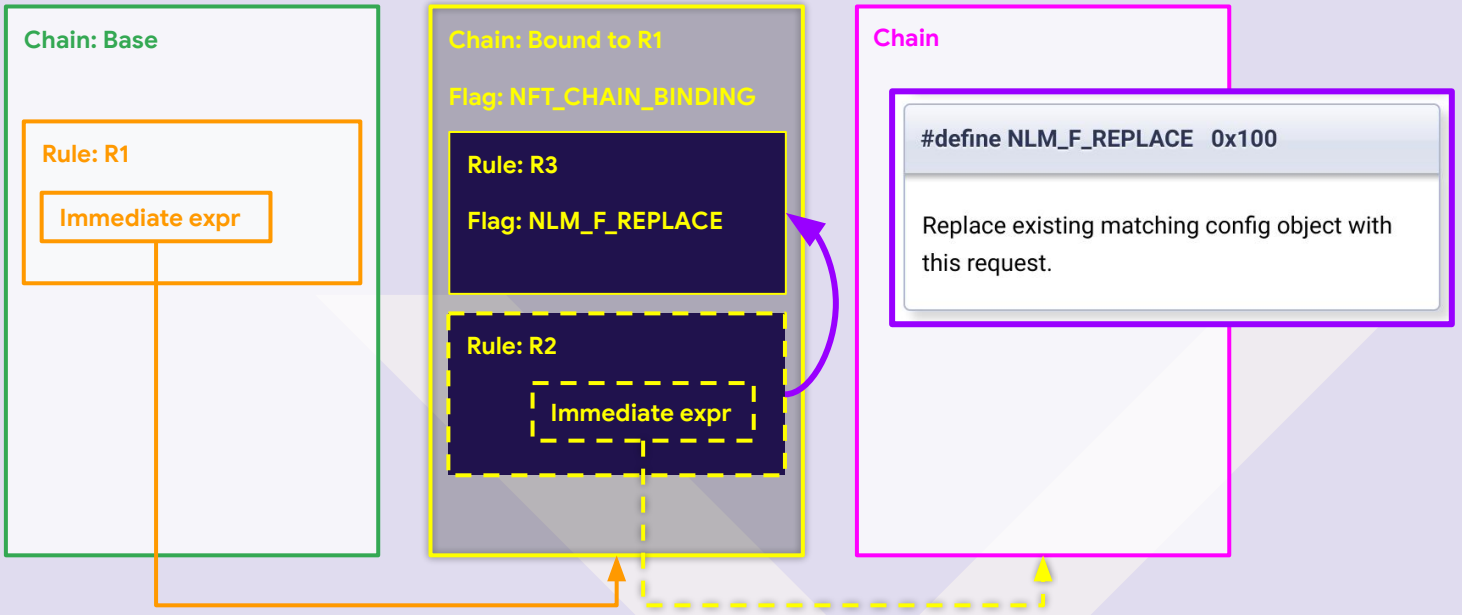
Send Exploits!

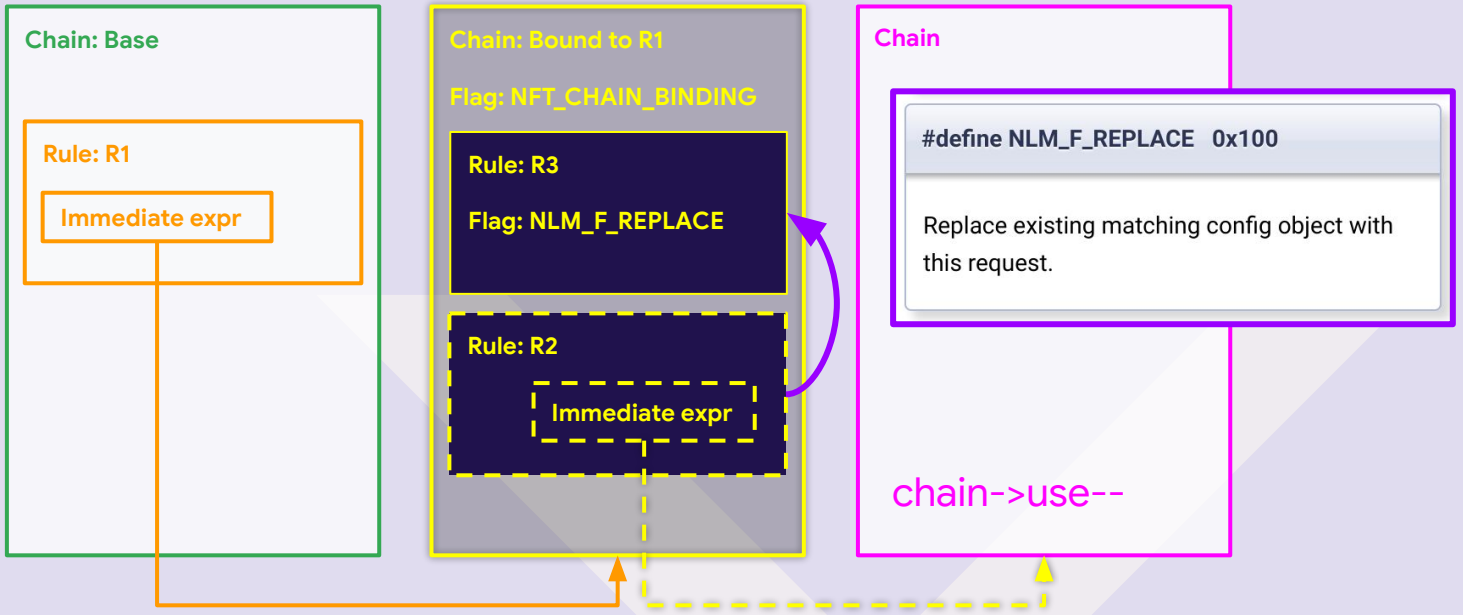
kCTF

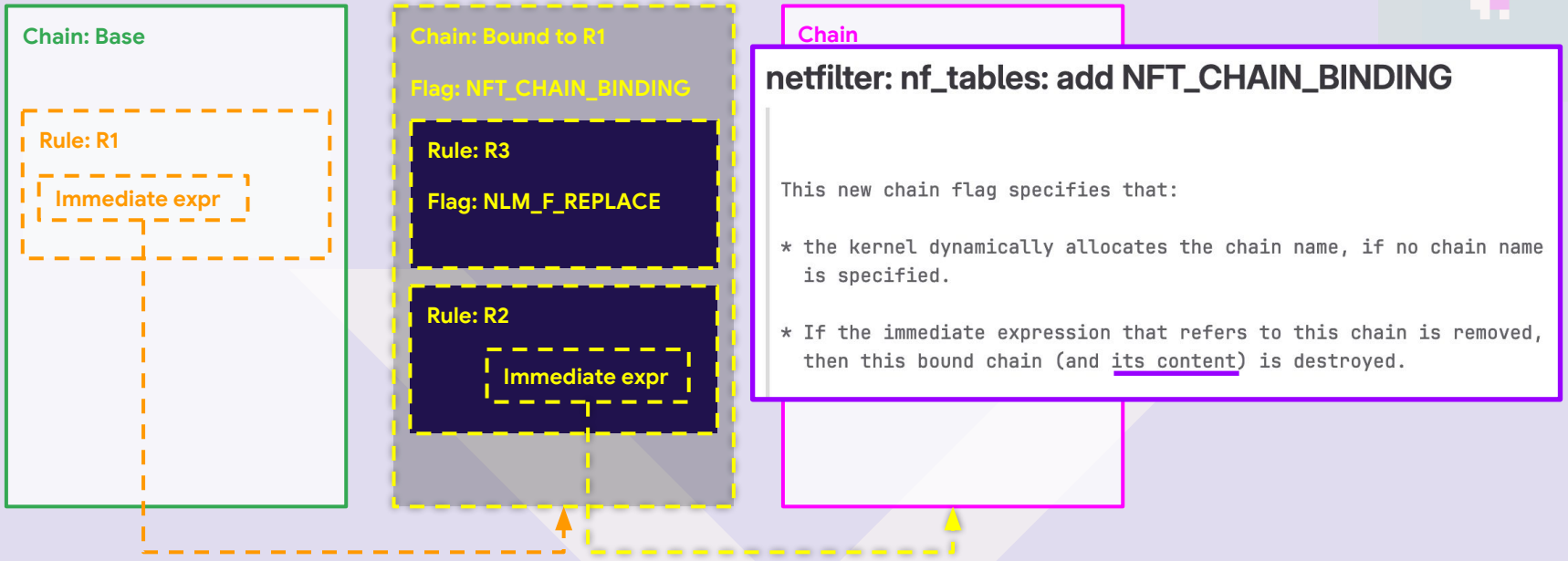
$1,205,200

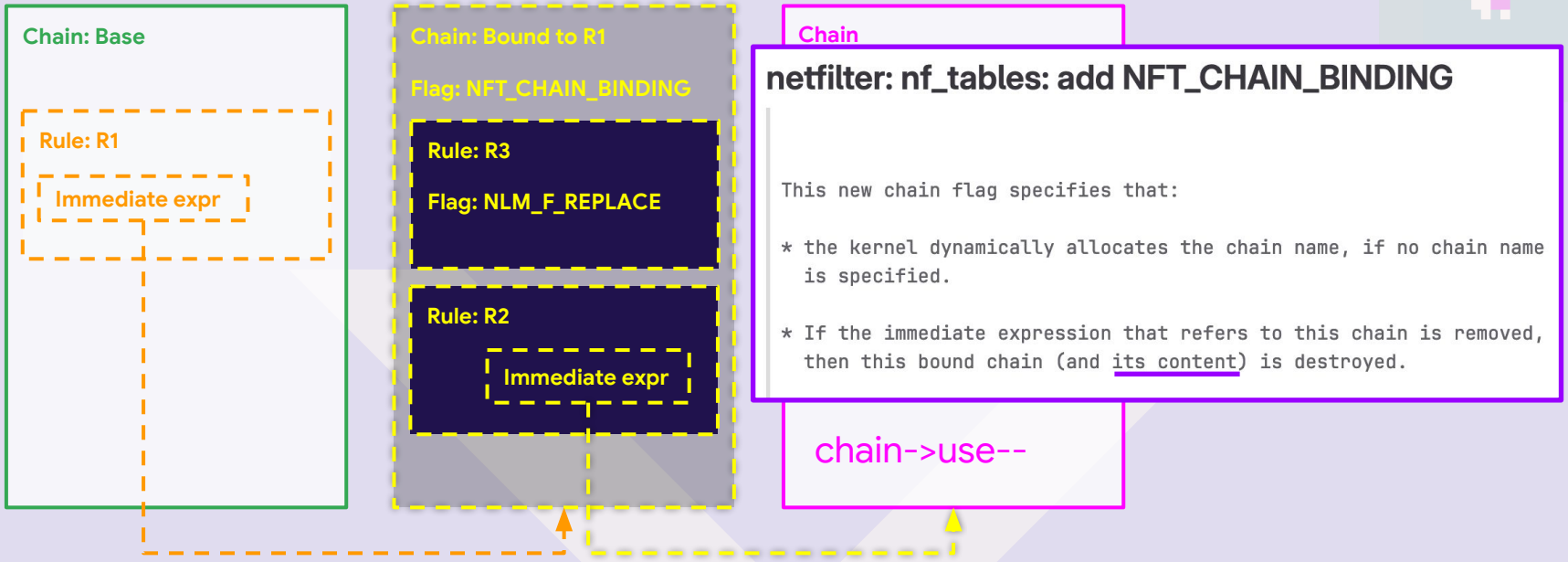

Chain: Base
Rule: R1
Immediate expr
Chain: Bound to R1
Flag: NFT_CHAIN_BINDING
Rule: R2
Immediate expr
Chain

NFTs


Chain: Base
Rule: R1
Immediate expr
Chain: Bound to R1
Flag: NFT_CHAIN_BINDING
Rule: R3
Flag: NLM_F_REPLACE
Rule: R2
Immediate expr
Chain
#define NLM_F_REPLACE 0x100
Replace existing matching config object with this request.


Chain: Base
Rule: R1
Immediate expr
Chain: Bound to R1
Flag: NFT_CHAIN_BINDING
Rule: R3
Flag: NLM_F_REPLACE
Rule: R2
Immediate expr
Chain
#define NLM_F_REPLACE 0x100
Replace existing matching config object with this request.
chain->use--

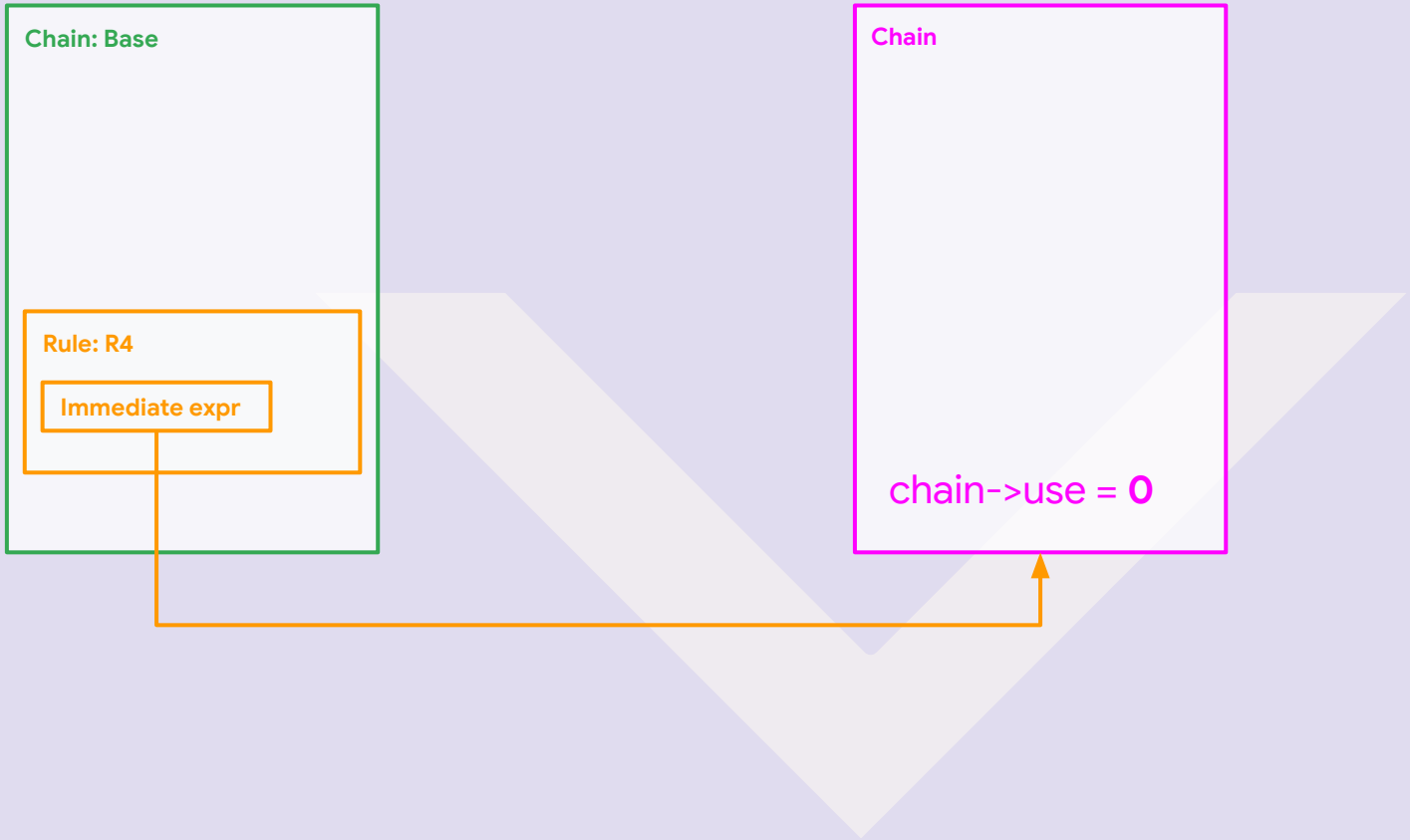

NFTs
Chain: Base
Rule: R1
Immediate expr
Chain: Bound to R1
Flag: NFT_CHAIN_BINDING
Rule: R3
Flag: NLM_F_REPLACE
Rule: R2
Immediate expr
Chain
netfilter: nf_tables: add NFT_CHAIN_BINDING
This new chain flag specifies that:
* the kernel dynamically allocates the chain name, if no chain name is specified.
* If the immediate expression that refers to this chain is removed, then this bound chain (and its content) is destroyed.

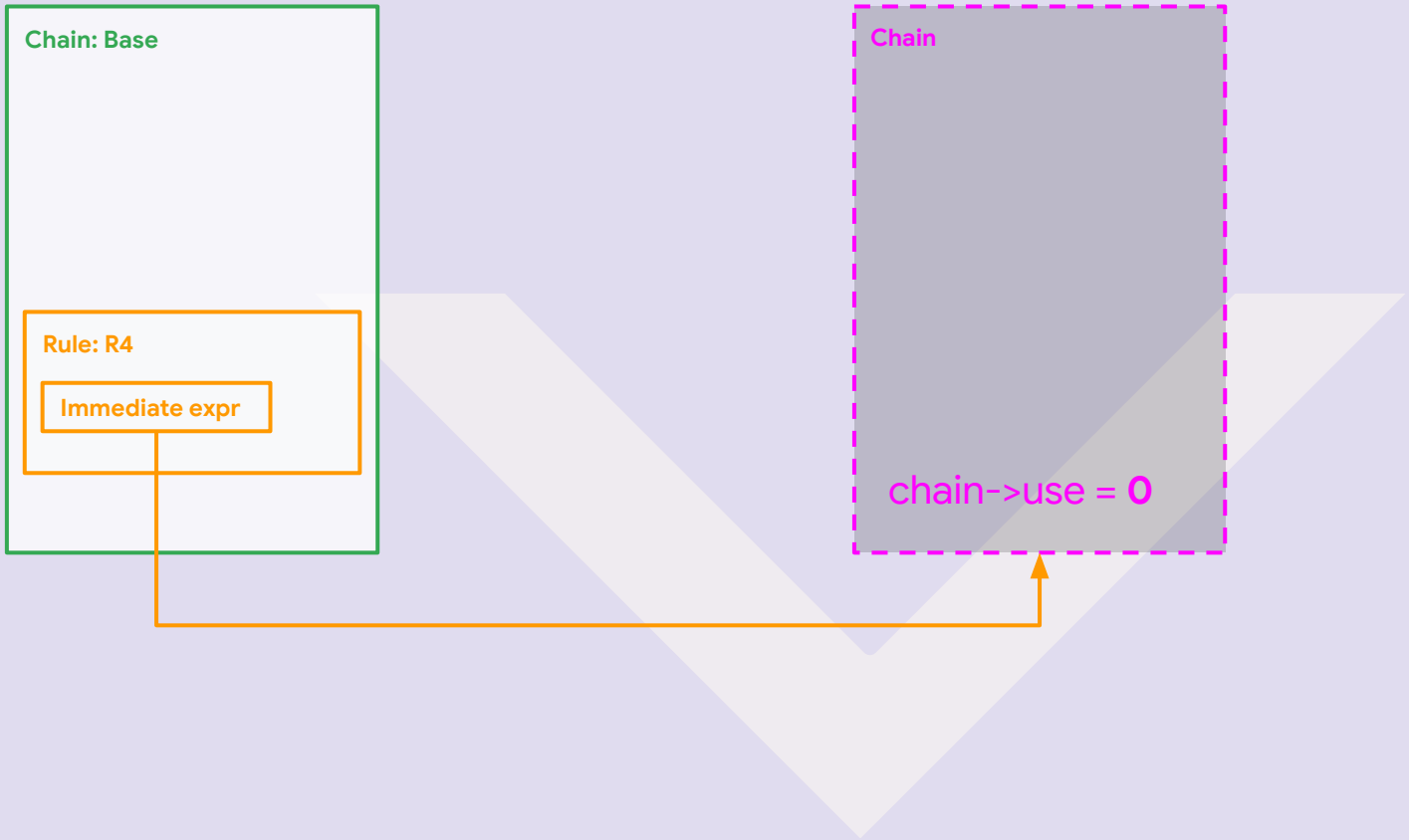

NFTs
Chain: Base
Rule: R1
Immediate expr
Chain: Bound to R1
Flag: NFT_CHAIN_BINDING
Rule: R3
Flag: NLM_F_REPLACE
Rule: R2
Immediate expr
Chain
netfilter: nf_tables: add NFT_CHAIN_BINDING
This new chain flag specifies that:
* the kernel dynamically allocates the chain name, if no chain name is specified.
* If the immediate expression that refers to this chain is removed, then this bound chain (and its content) is destroyed.
chain->use--

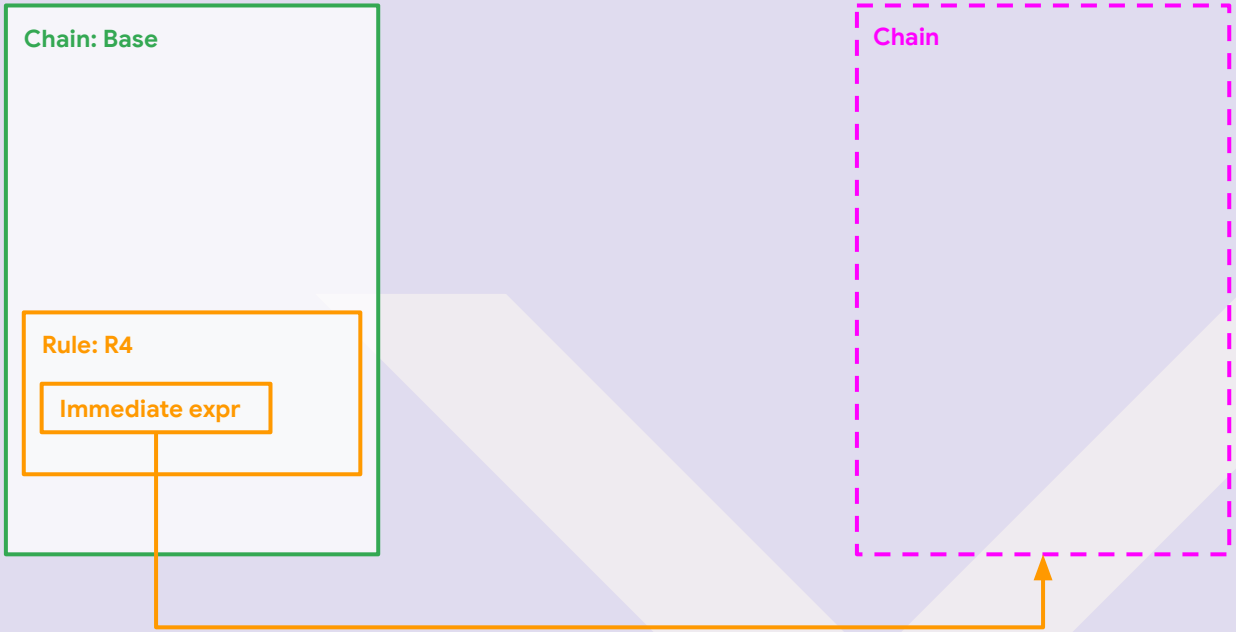
Chain: Base

Chain

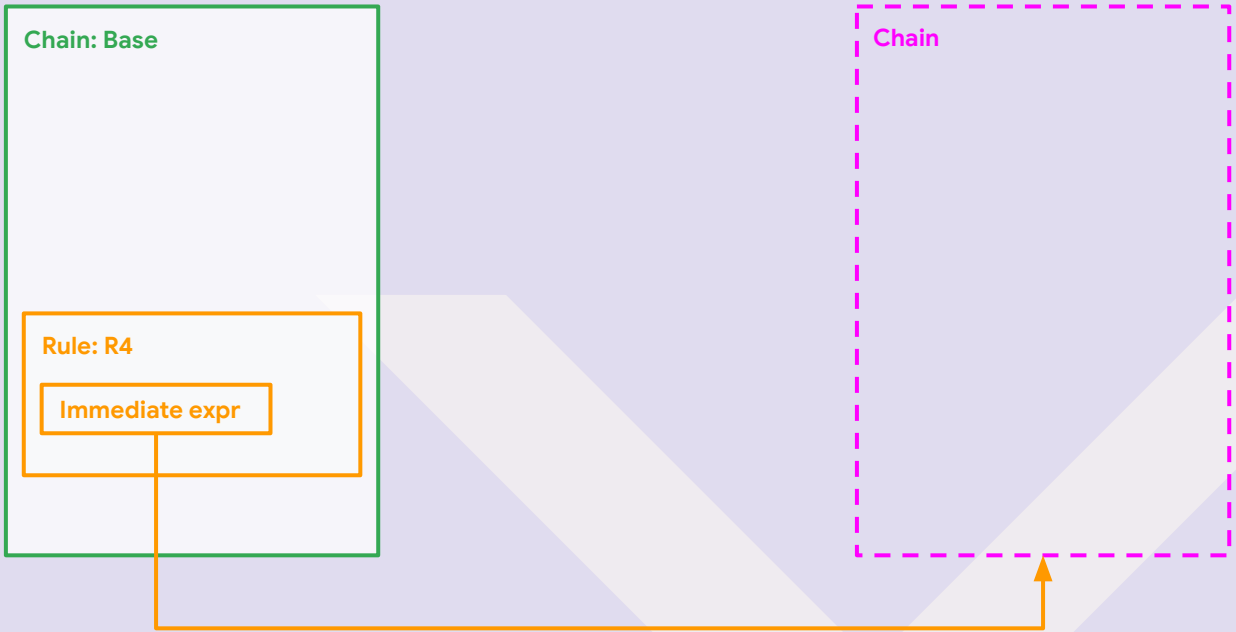
chain->use = **-1**


NFTs
Chain: Base
Rule: R4
Immediate expr
Chain
chain->use = 0


NFTs
Chain: Base
Rule: R4
Immediate expr
Chain
chain->use = 0



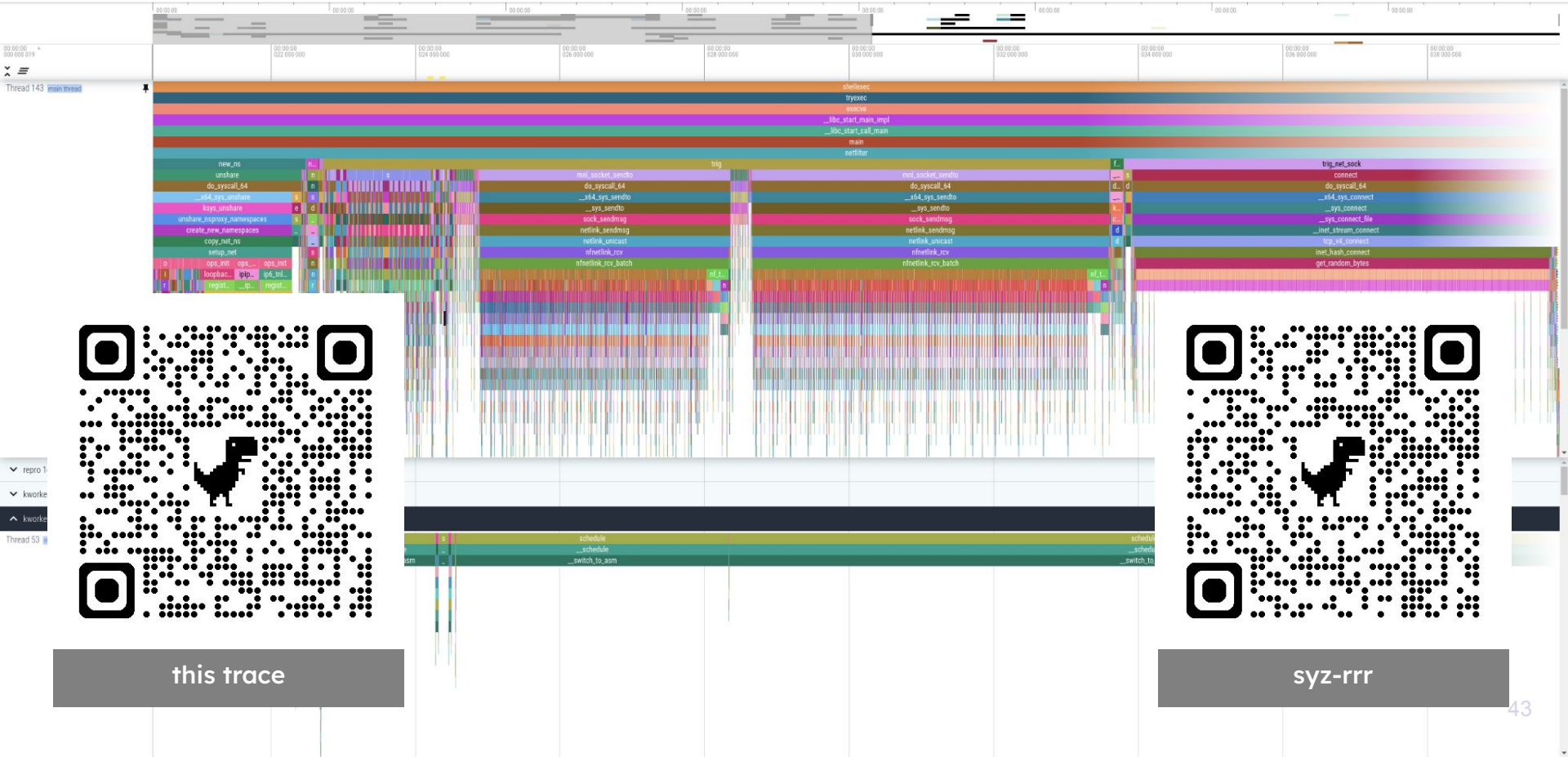
Rule can still use Chain after it was kfree()'d



Rule can still **use** Chain **after** it was k**free**()'d

Can this "**use-after-free**" be used for privilege escalation?

10,000 feet view of a typical exploit

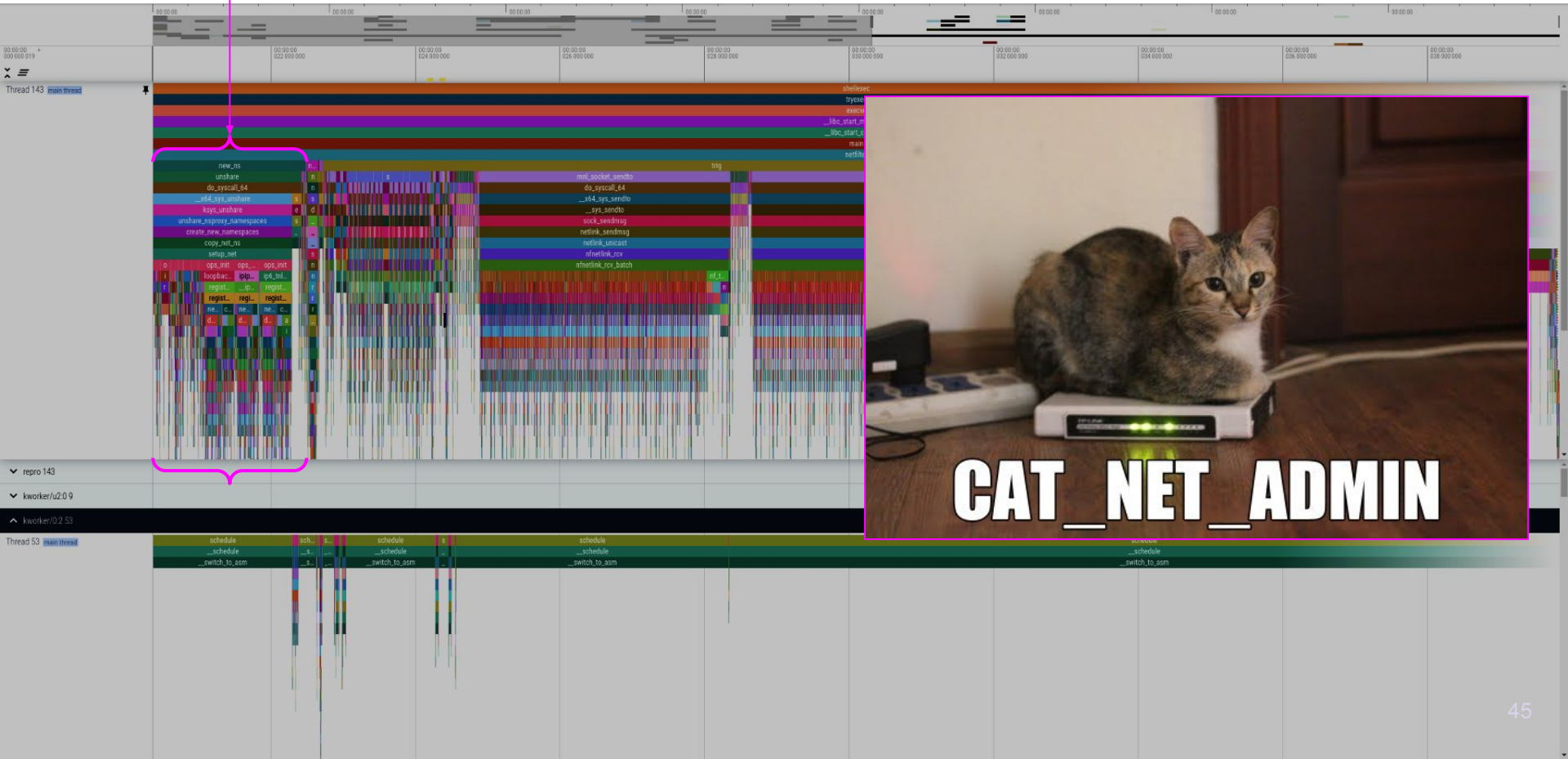


10,000 feet view of a typical exploit

Artem Metla

Florent Revest

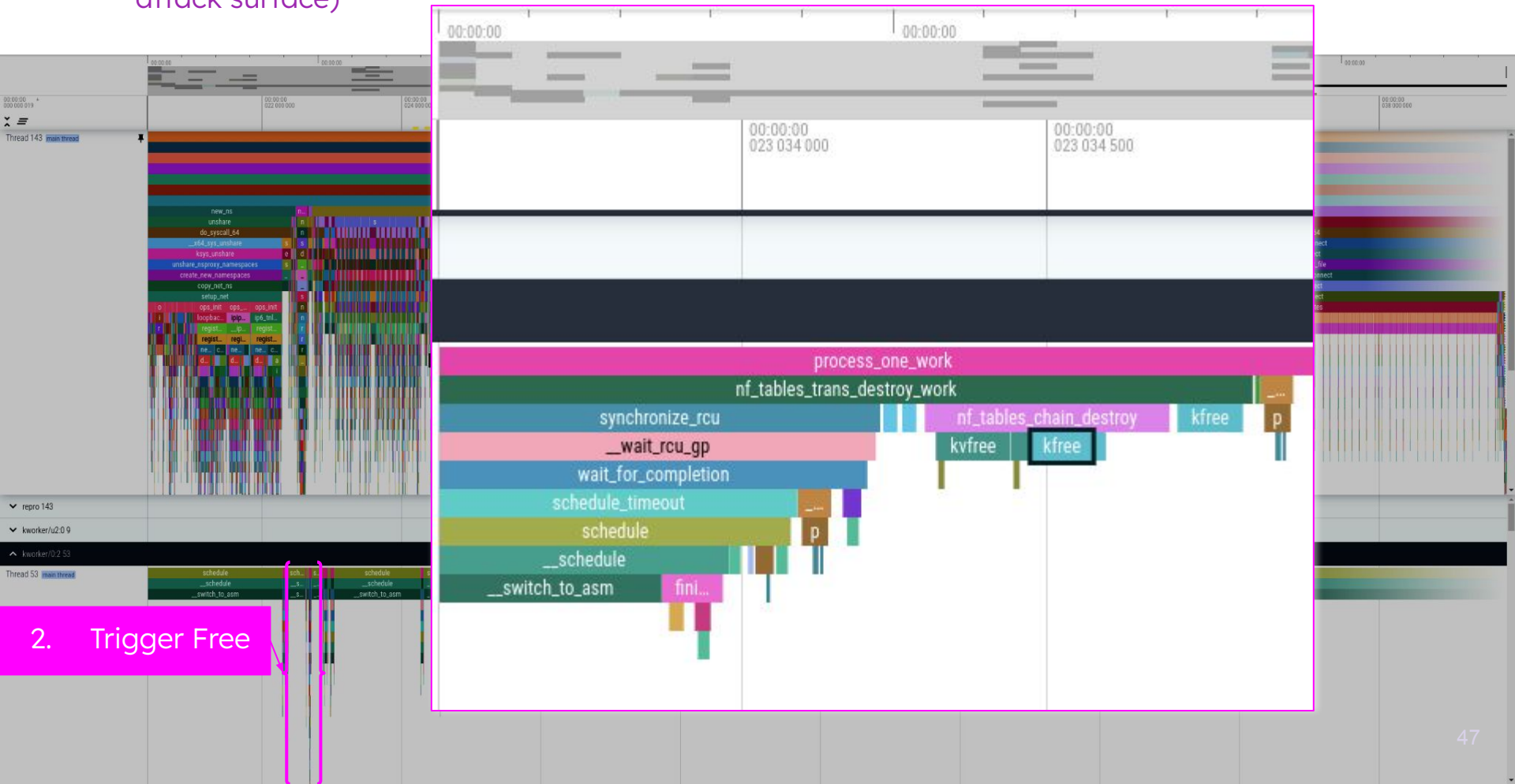
1. New NS (to reach attack surface)



1. New NS (to reach attack surface)

2. Trigger Free

1. New NS (to reach attack surface)



2. Trigger Free

1. New NS (to reach attack surface)

net / netfilter / nf_tables_api.c

```
void nf_tables_chain_destroy(struct nft_ctx *ctx)
{
        struct nft_chain *chain = ctx->chain;
        struct nft_hook *hook, *next;

        if (WARN_ON(chain->use > 0))
                return;

        /* no concurrent access possible anymore */
        nf_tables_chain_free_chain_rules(chain);

        if (nft_is_base_chain(chain)) {
                struct nft_base_chain *basechain = nft_base_chain(chain);

                if (nft_base_chain_netdev(ctx->family, basechain->ops.hooknum)) {
                        list_for_each_entry_safe(hook, next,
                                                 &basechain->hook_list, list) {
                                list_del_rcu(&hook->list);
                                kfree_rcu(hook, rcu);
                        }
                }
                module_put(basechain->type->owner);
                if (rcu_access_pointer(basechain->stats)) {
                        static_branch_dec(&nft_counters_enabled);
                        free_percpu(rcu_dereference_raw(basechain->stats));
                }
                kfree(chain->name);
                kfree(chain->udata);
                kfree(basechain);
        } else {
                kfree(chain->name);
                kfree(chain->udata);
                kfree(chain);
        }
}
```

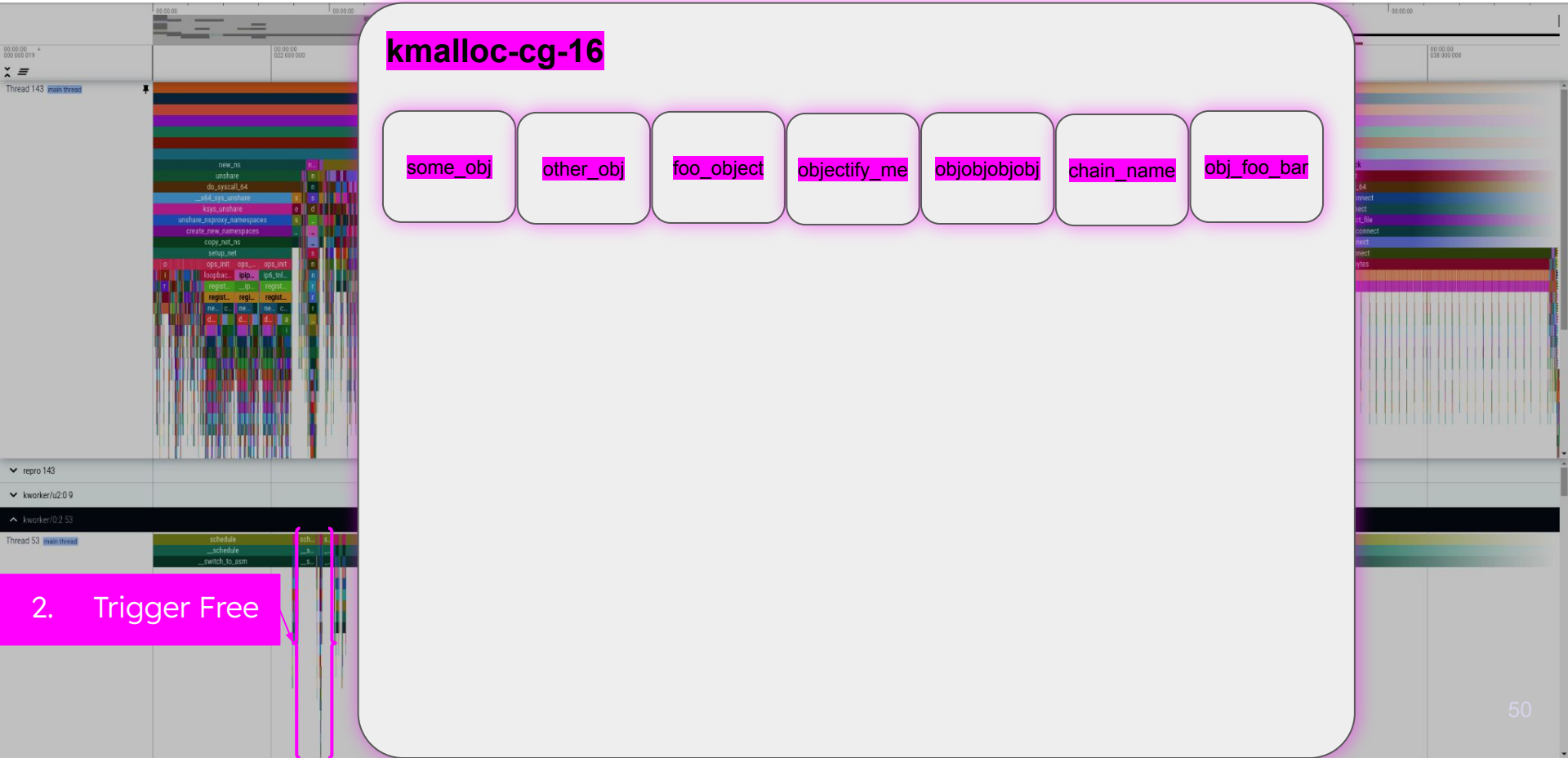
synchronize_rcu
__wait_rcu_gp
wait_for_completion
schedule_timeout
schedule
__schedule
__switch_to_asm
fini...
nf_tables_chain_destroy
kfree
kvfree
kfree

2. Trigger Free

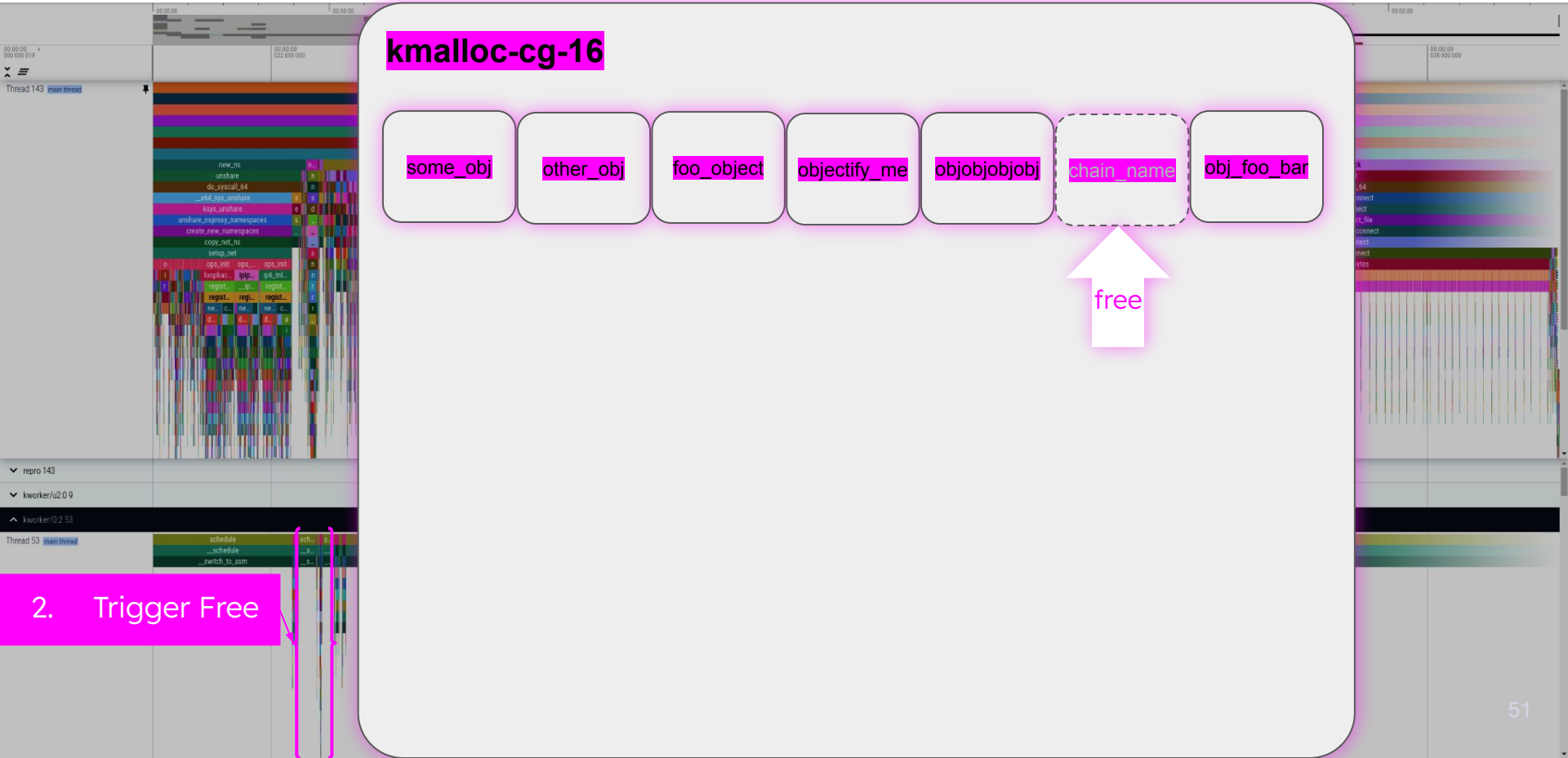
1. New NS (to reach attack surface)

2. Trigger Free

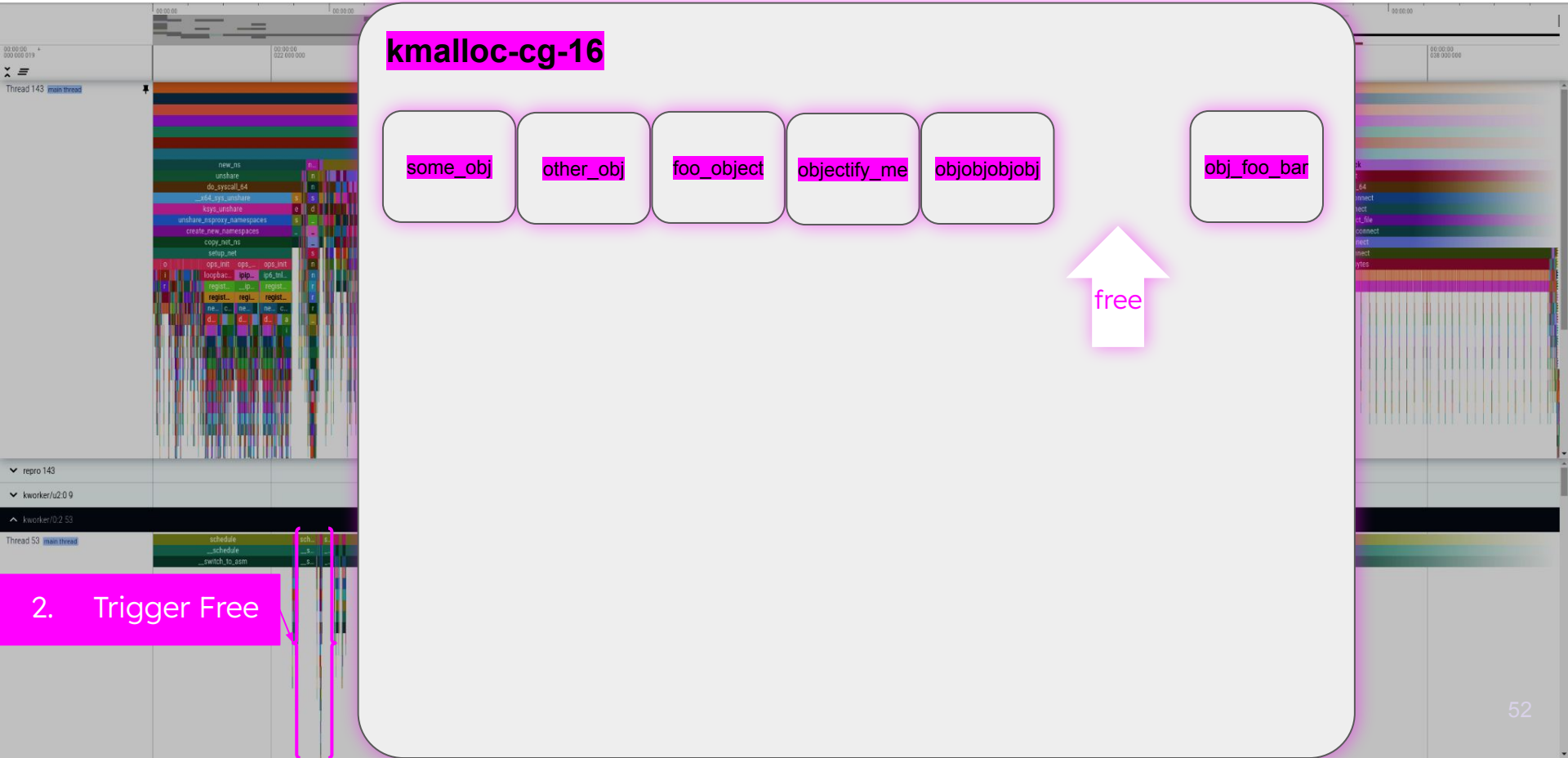
1. New NS (to reach attack surface)



1. New NS (to reach attack surface)



1. New NS (to reach attack surface)



1. New NS (to reach attack surface)

3. Spray object

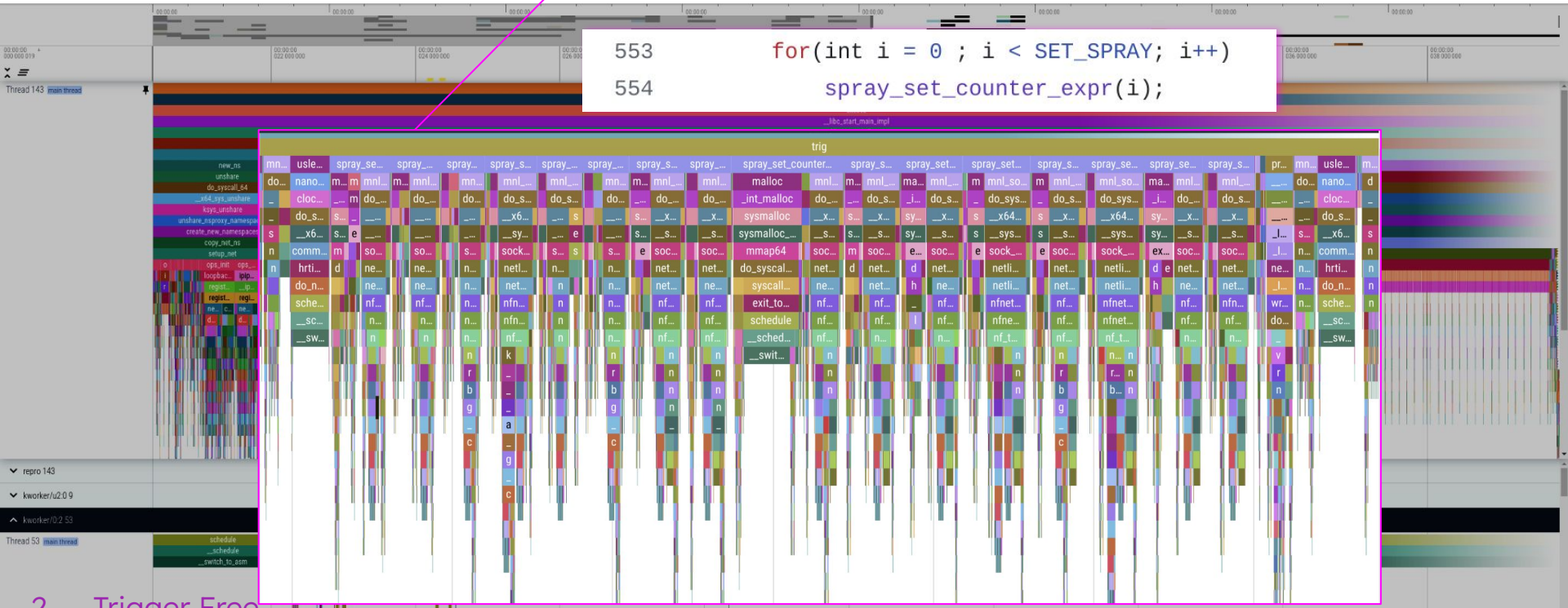
```
553        for(int i = 0 ; i < SET_SPRAY; i++)
554            spray_set_counter_expr(i);
```

2. Trigger Free

1. New NS (to reach attack surface)

3. Spray object

```
553        for(int i = 0 ; i < SET_SPRAY; i++)
554            spray_set_counter_expr(i);
```

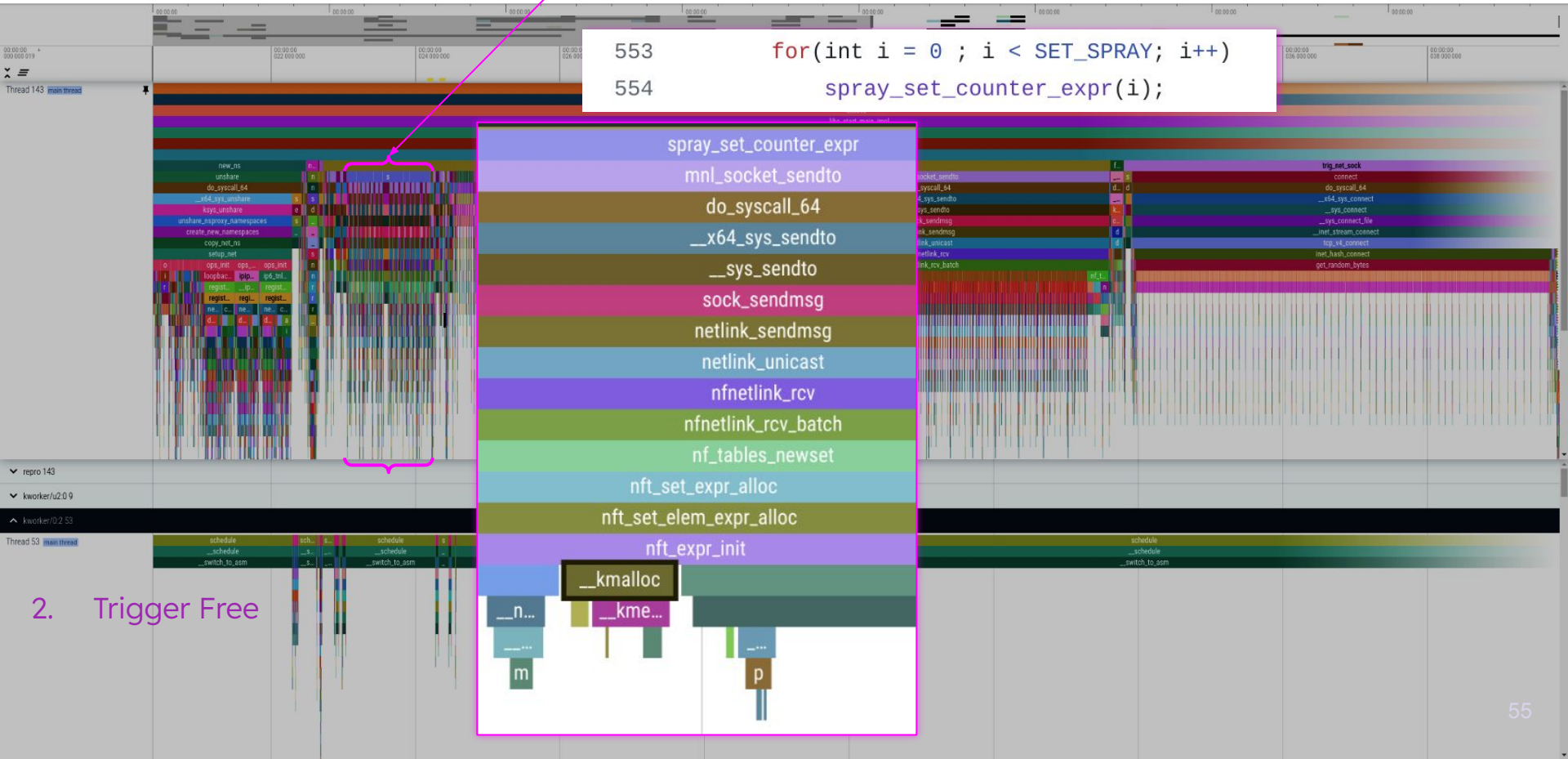


2. Trigger Free

1. New NS (to reach attack surface)

3. Spray object

```
553         for(int i = 0 ; i < SET_SPRAY; i++)
554             spray_set_counter_expr(i);
```

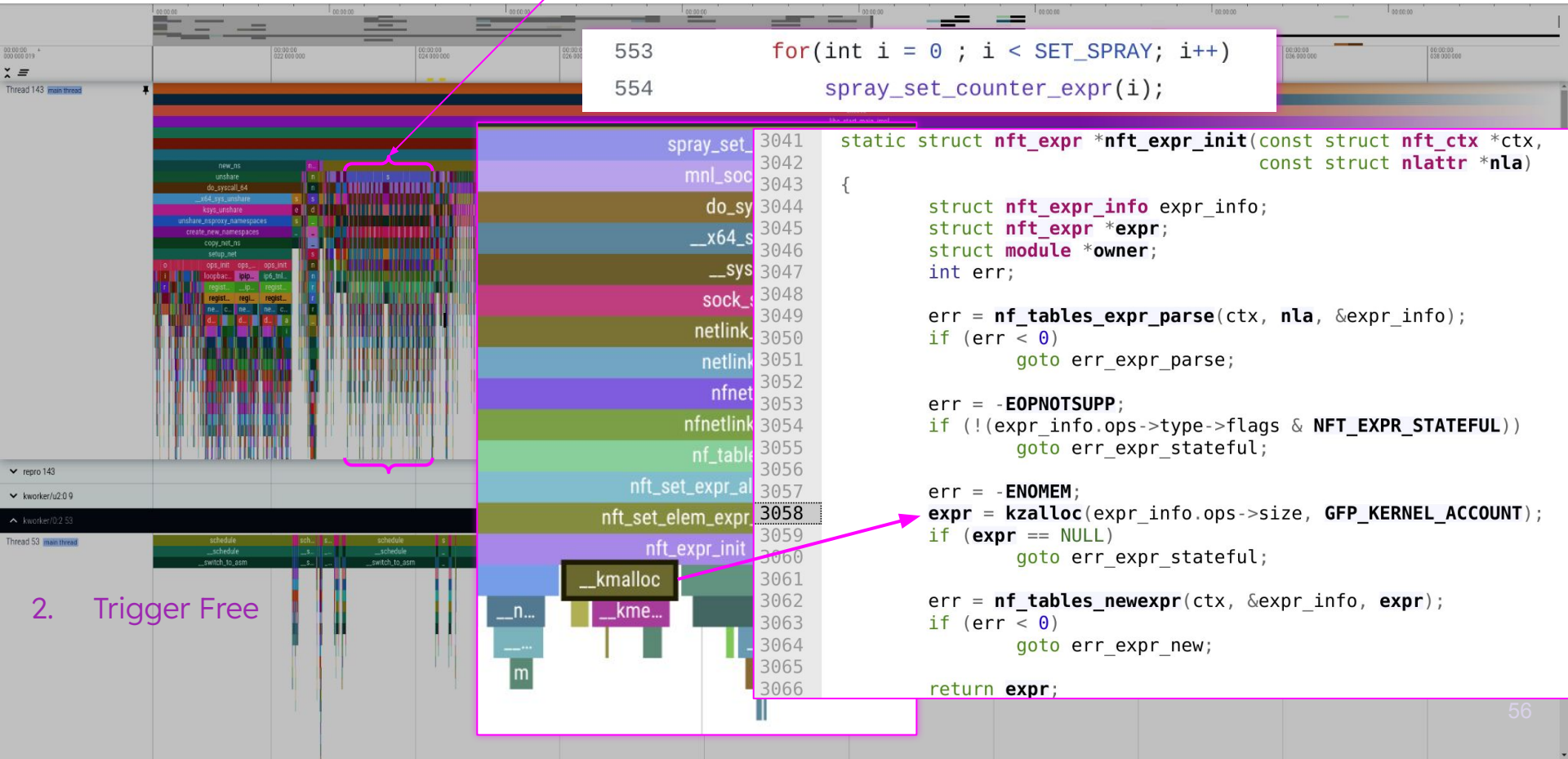


2. Trigger Free

1. New NS (to reach attack surface)

3. Spray object

```
553         for(int i = 0 ; i < SET_SPRAY; i++)
554             spray_set_counter_expr(i);
```




```
3041  static struct nft_expr *nft_expr_init(const struct nft_ctx *ctx,
3042                                        const struct nlattr *nla)
3043  {
3044          struct nft_expr_info expr_info;
3045          struct nft_expr *expr;
3046          struct module *owner;
3047          int err;
3048
3049          err = nf_tables_expr_parse(ctx, nla, &expr_info);
3050          if (err < 0)
3051                  goto err_expr_parse;
3052
3053          err = -EOPNOTSUPP;
3054          if (!(expr_info.ops->type->flags & NFT_EXPR_STATEFUL))
3055                  goto err_expr_stateful;
3056
3057          err = -ENOMEM;
3058          expr = kzalloc(expr_info.ops->size, GFP_KERNEL_ACCOUNT);
3059          if (expr == NULL)
3060                  goto err_expr_stateful;
3061
3062          err = nf_tables_newexpr(ctx, &expr_info, expr);
3063          if (err < 0)
3064                  goto err_expr_new;
3065
3066          return expr;
```

2. Trigger Free

1. New NS (to reach attack surface)

3. Spray object

```
553         for(int i = 0 ; i < SET_SPRAY; i++)
554             spray_set_counter_expr(i);
```

kmalloc-cg-16

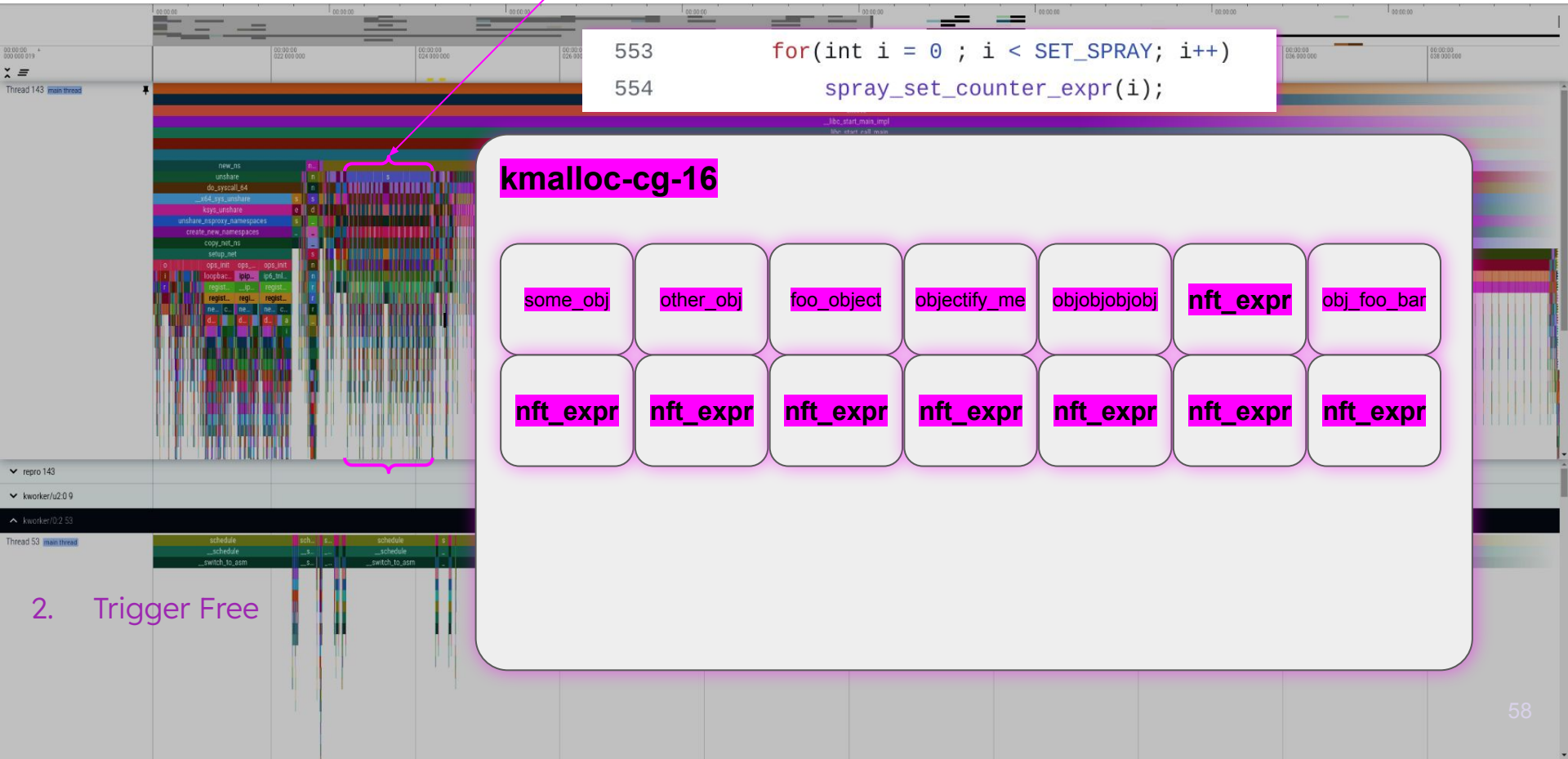
some_obj | other_obj | foo_object | objectify_me | objobjobjobj | obj_foo_bar

2. Trigger Free

1. New NS (to reach attack surface)

3. Spray object

```
553         for(int i = 0 ; i < SET_SPRAY; i++)
554             spray_set_counter_expr(i);
```

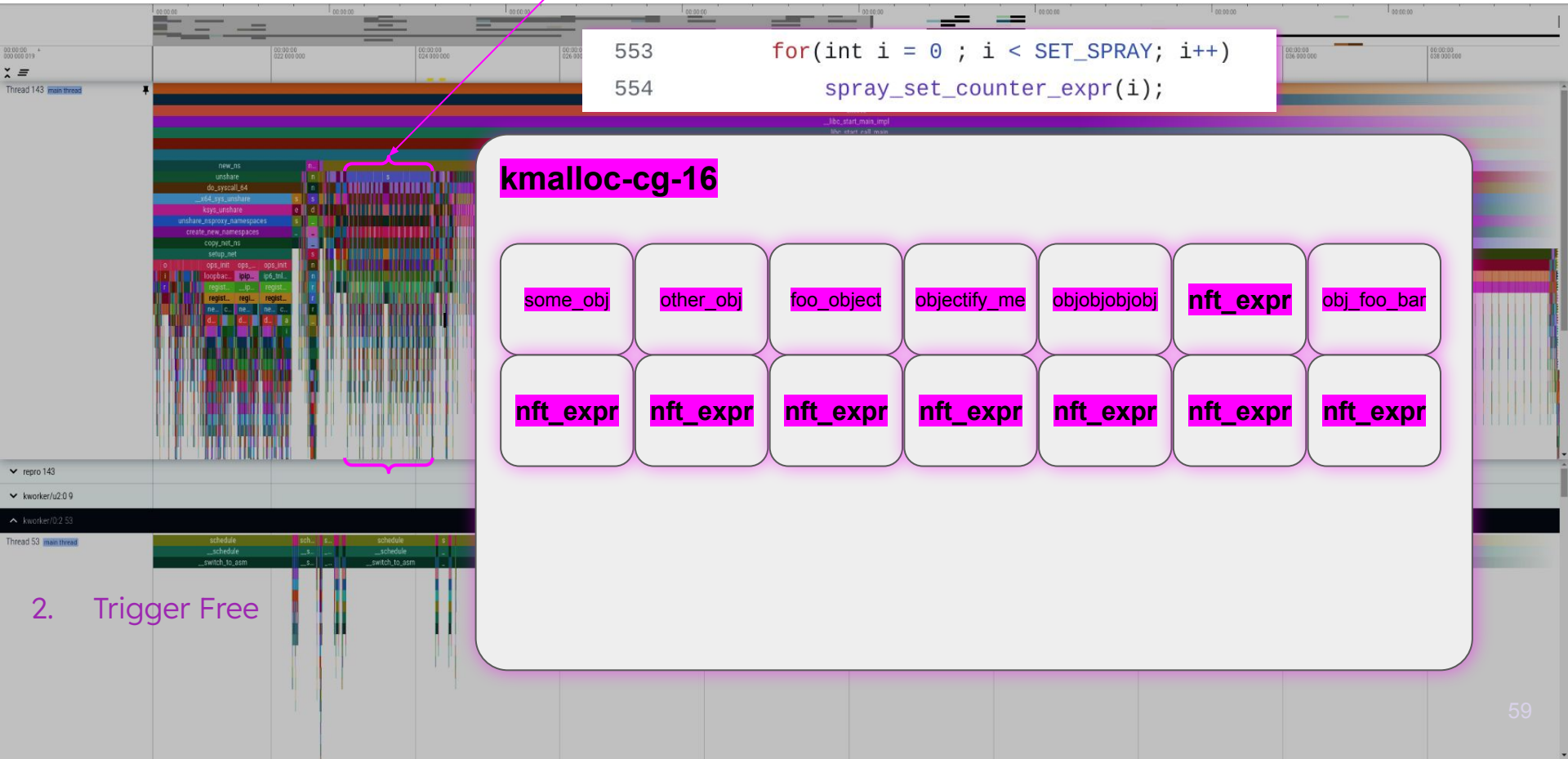


2. Trigger Free

1. New NS (to reach attack surface)

3. Spray object

```
553         for(int i = 0 ; i < SET_SPRAY; i++)
554             spray_set_counter_expr(i);
```



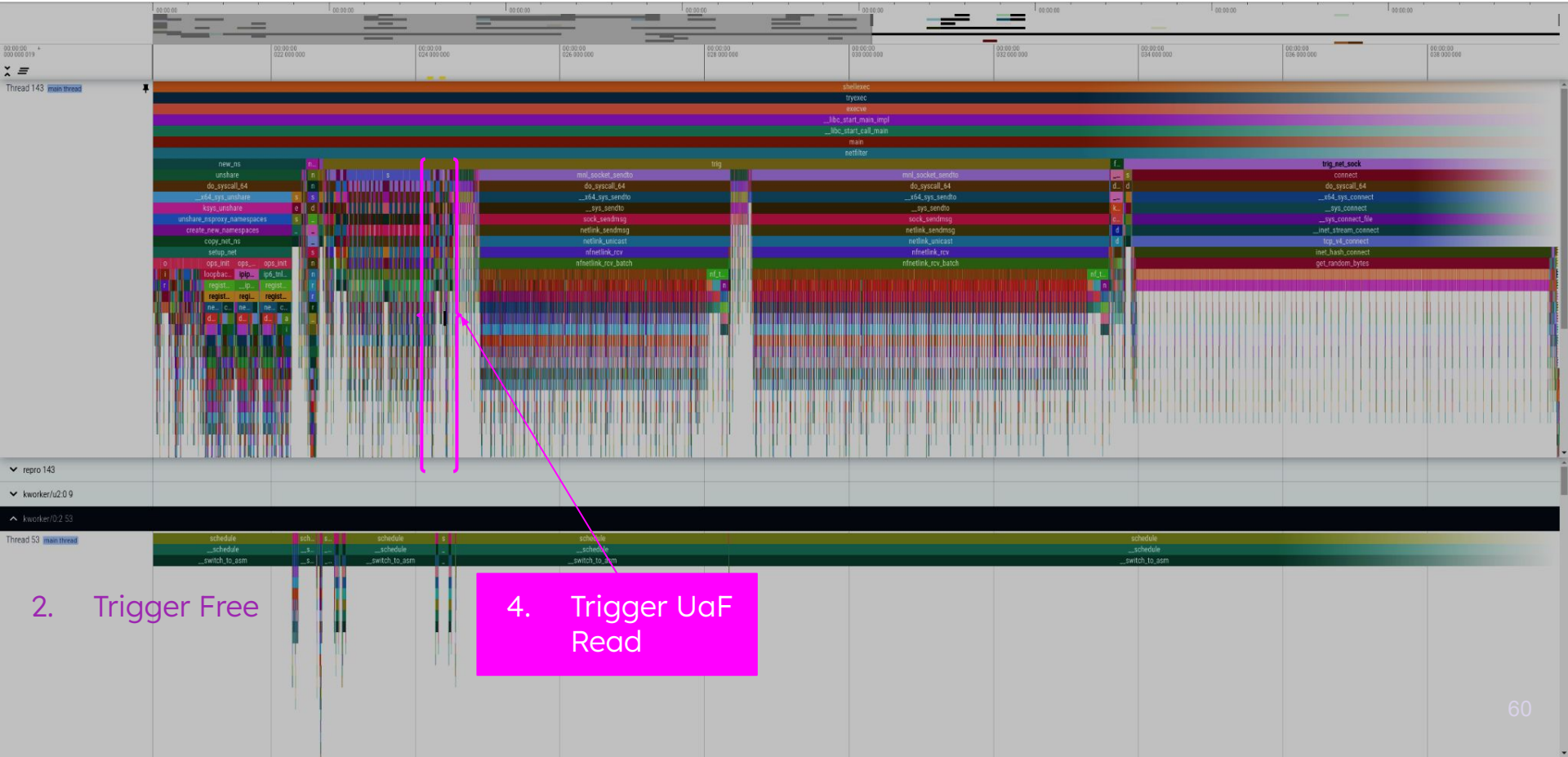
kmalloc-cg-16

some_obj	other_obj	foo_object	objectify_me	objobjobjobj	**nft_expr**	obj_foo_bar
nft_expr	**nft_expr**	**nft_expr**	**nft_expr**	**nft_expr**	**nft_expr**	**nft_expr**

2. Trigger Free

1. New NS (to reach attack surface)

3. Spray object



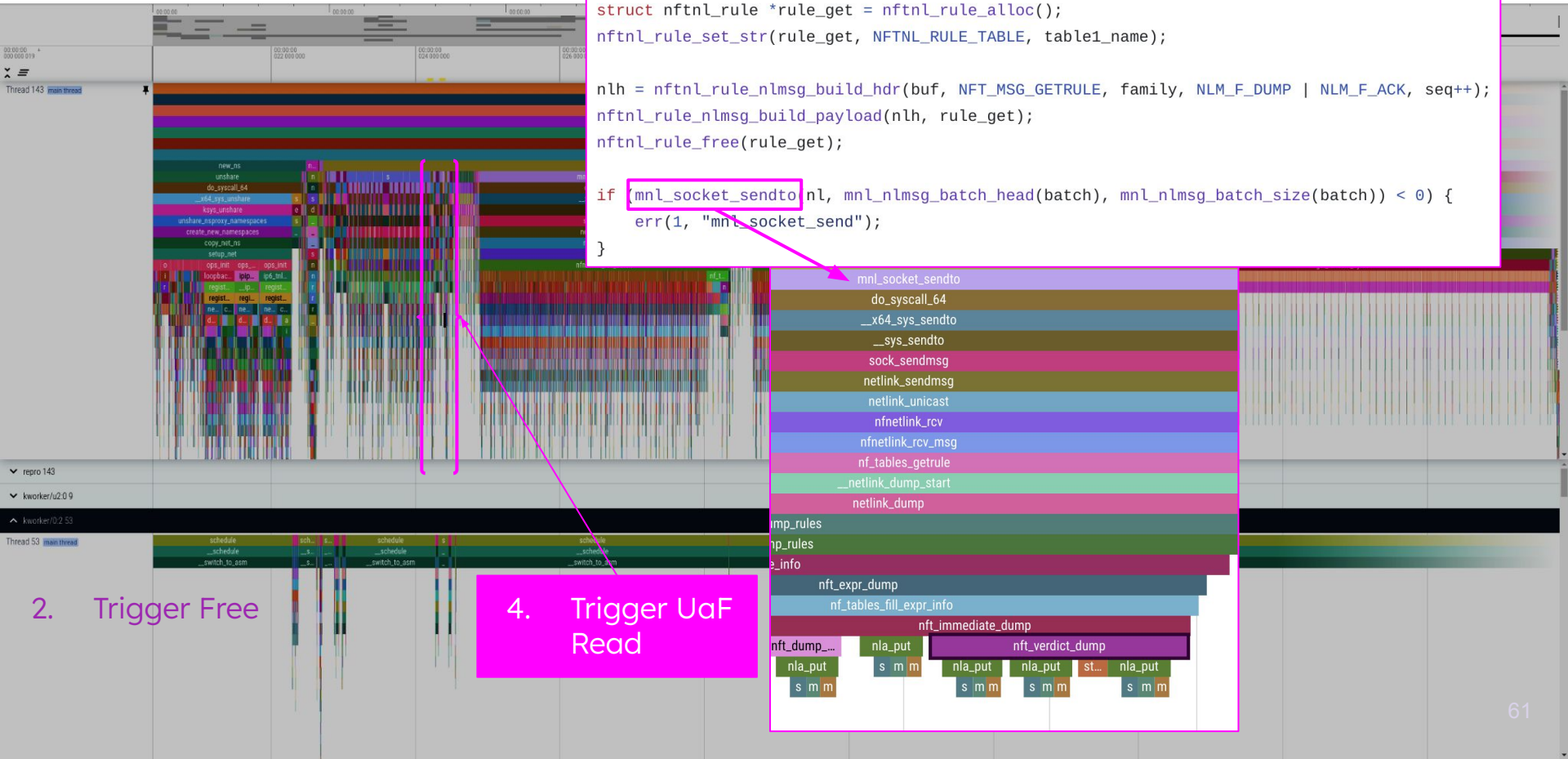
2. Trigger Free

4. Trigger UaF Read

1. New NS (to reach attack surface)

3. Spray object

```
struct nftnl_rule *rule_get = nftnl_rule_alloc();
nftnl_rule_set_str(rule_get, NFTNL_RULE_TABLE, table1_name);

nlh = nftnl_rule_nlmsg_build_hdr(buf, NFT_MSG_GETRULE, family, NLM_F_DUMP | NLM_F_ACK, seq++);
nftnl_rule_nlmsg_build_payload(nlh, rule_get);
nftnl_rule_free(rule_get);

if (mnl_socket_sendto(nl, mnl_nlmsg_batch_head(batch), mnl_nlmsg_batch_size(batch)) < 0) {
    err(1, "mnl_socket_send");
}
```



2. Trigger Free

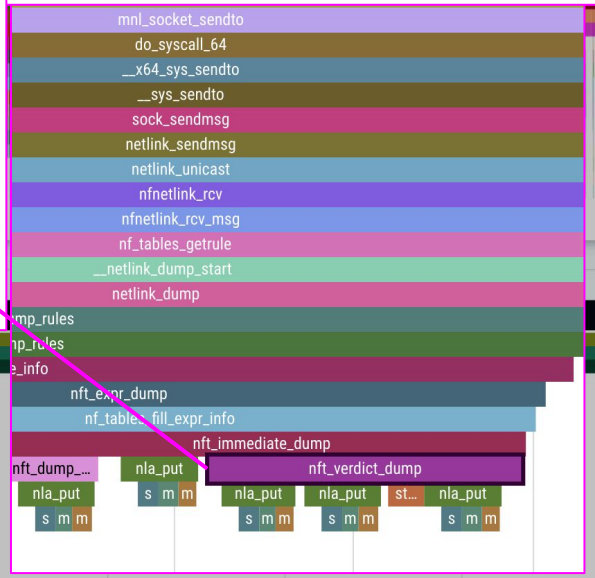
4. Trigger UaF Read

1. New NS (to reach attack surface)

3. Spray object

```c
struct nftnl_rule *rule_get = nftnl_rule_alloc();
nftnl_rule_set_str(rule_get, NFTNL_RULE_TABLE, table1_name);

lmsg_build_hdr(buf, NFT_MSG_GETRULE, family, NLM_F_DUMP | NLM_F_ACK, seq++);
uild_payload(nlh, rule_get);
le_get);

dto(nl, mnl_nlmsg_batch_head(batch), mnl_nlmsg_batch_size(batch)) < 0) {
cket_send");
```

```c
int nft_verdict_dump(struct sk_buff *skb, int type, const struct nft_verdict *v)
{
        struct nlattr *nest;

        nest = nla_nest_start_noflag(skb, type);
        if (!nest)
                goto nla_put_failure;

        if (nla_put_be32(skb, NFTA_VERDICT_CODE, htonl(v->code)))
                goto nla_put_failure;

        switch (v->code) {
        case NFT_JUMP:
        case NFT_GOTO:
                if (nla_put_string(skb, NFTA_VERDICT_CHAIN,
                                   v->chain->name))
                        goto nla_put_failure;
        }
        nla_nest_end(skb, nest);
        return 0;

nla_put_failure:
        return -1;
}
```



2. Trigger Free

4. Trigger UaF Read

1. New NS (to reach attack surface)

3. Spray object

```
int nft_verdict_dump(struct sk_buff *skb, int type, const struct nft_verdict *v)
{
        struct nlattr *nest;

        nest = nla_nest_start_noflag(skb, type);
        if (!nest)
                goto nla_put_failure;

        if (nla_put_be32(skb, NFTA_VERDICT_CODE, htonl(v->code)))
                goto nla_put_failure;

        switch (v->code) {
        case NFT_JUMP:
        case NFT_GOTO:
                if (nla_put_string(skb, NFTA_VERDICT_CHAIN,
                                   v->chain->name))
                        goto nla_put_failure;
        }
        nla_nest_end(skb, nest);
        return 0;

nla_put_failure:
        return -1;
}
```

kmalloc-cg-16

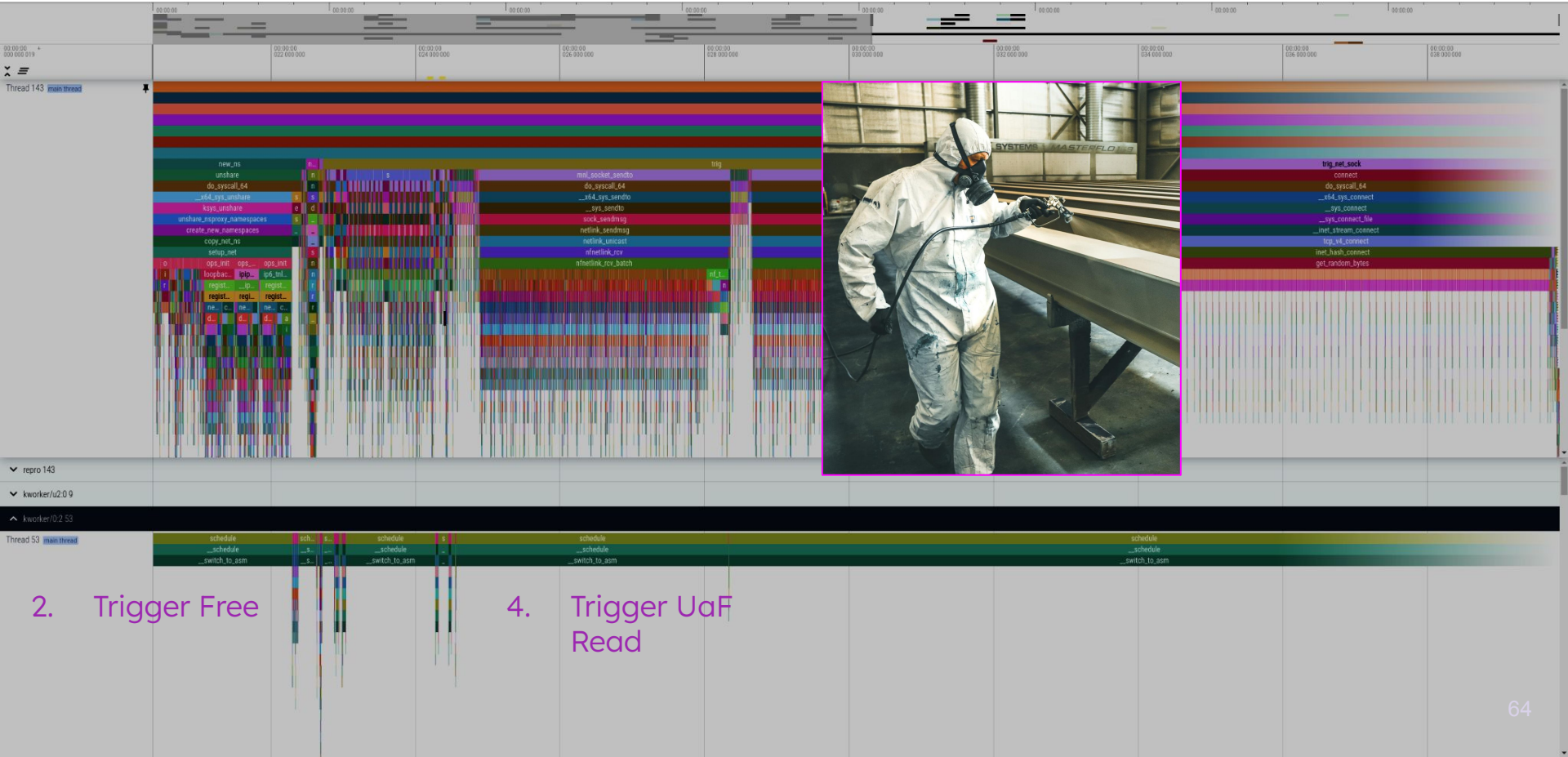
chain_name
+
nft_expr

```
struct nft_expr {
        const struct nft_expr_ops       *ops;
        unsigned char                   data[]
                __attribute__((aligned(__align
};
```

2. Trigger Free

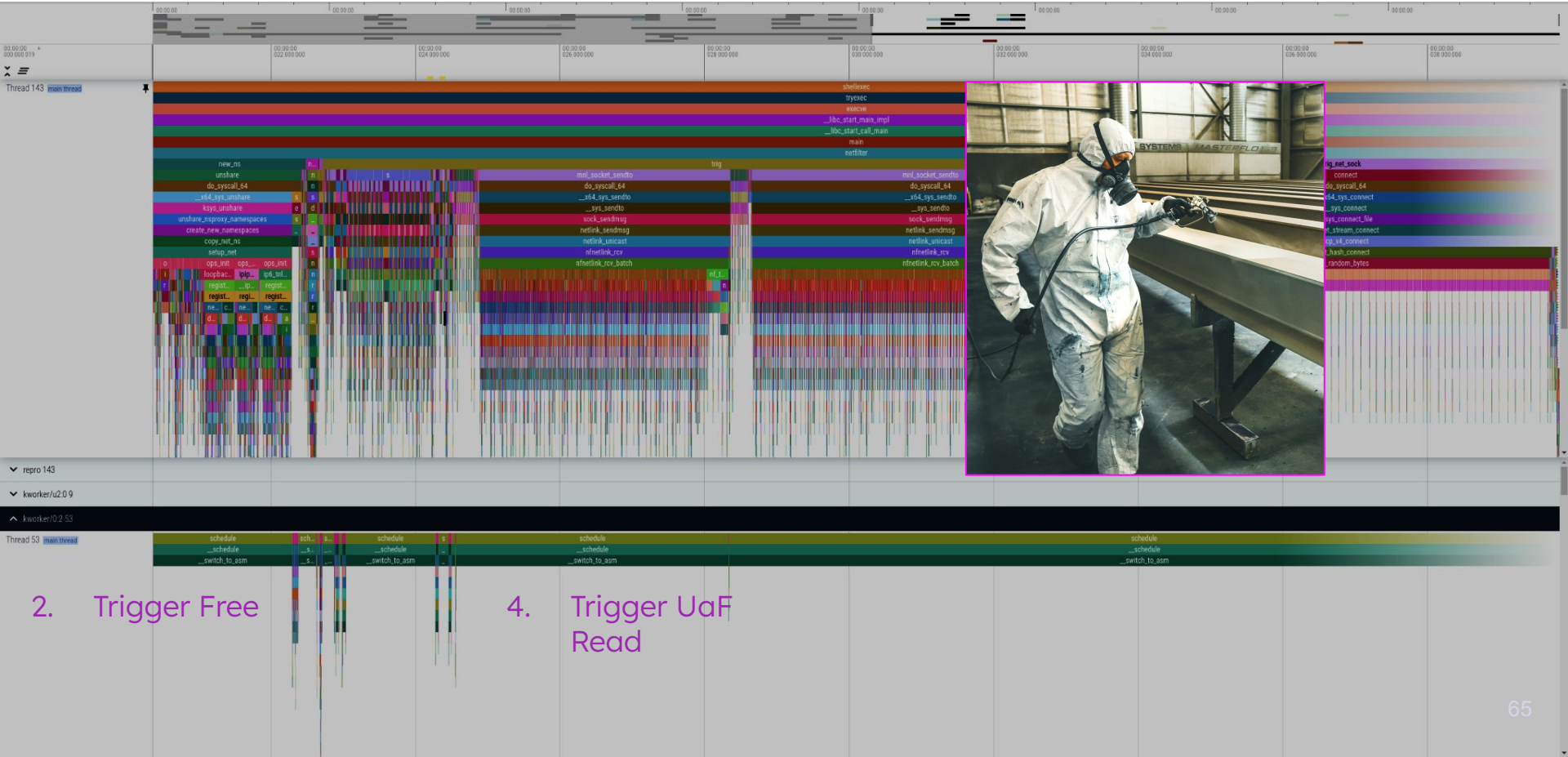
4. Trigger UaF Read

1. New NS (to reach attack surface)
3. Spray object
5. Spray object



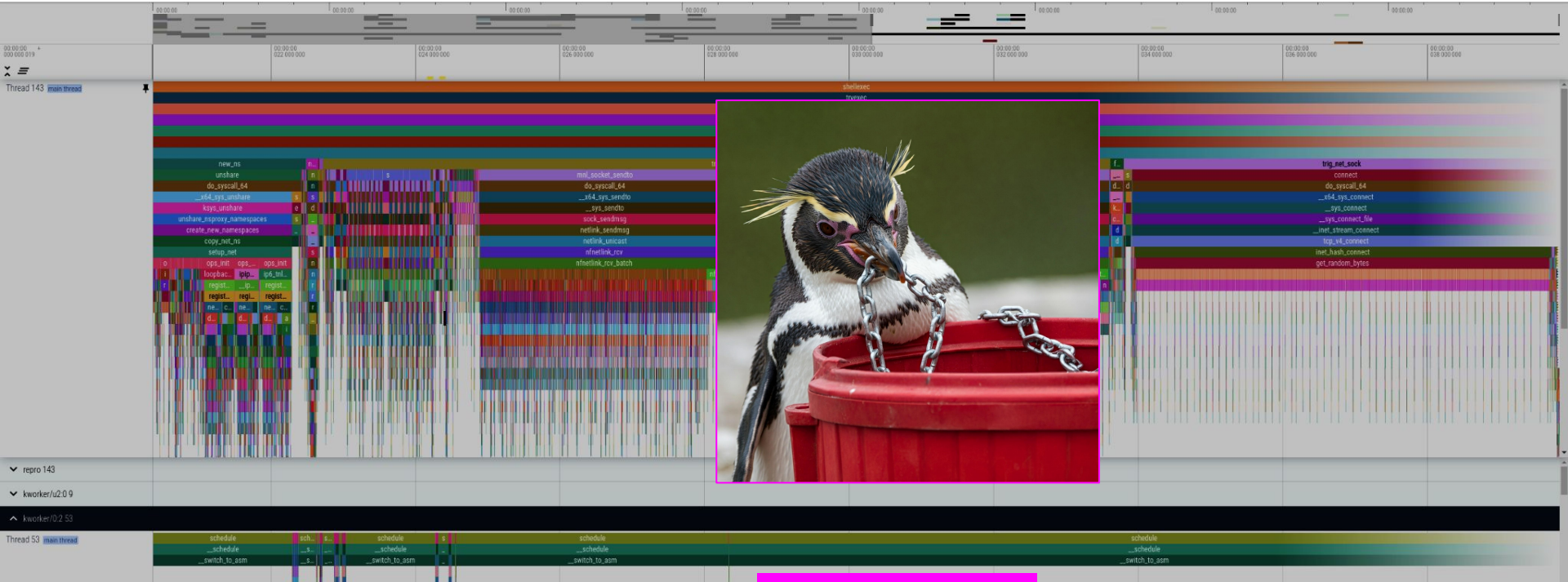
2. Trigger Free
4. Trigger UaF Read

1. New NS (to reach attack surface)
3. Spray object
5. Spray object
6. Spray object



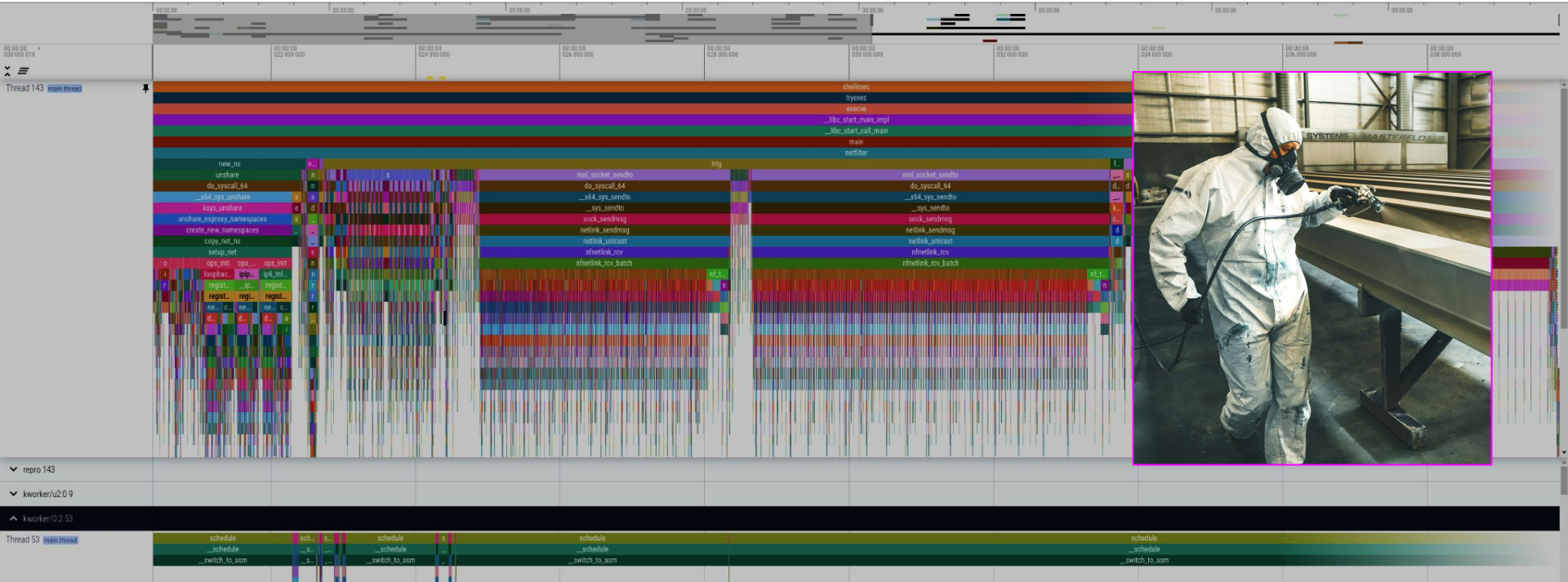
2. Trigger Free
4. Trigger UaF Read

1. New NS (to reach attack surface)
3. Spray object
5. Spray object
6. Spray object



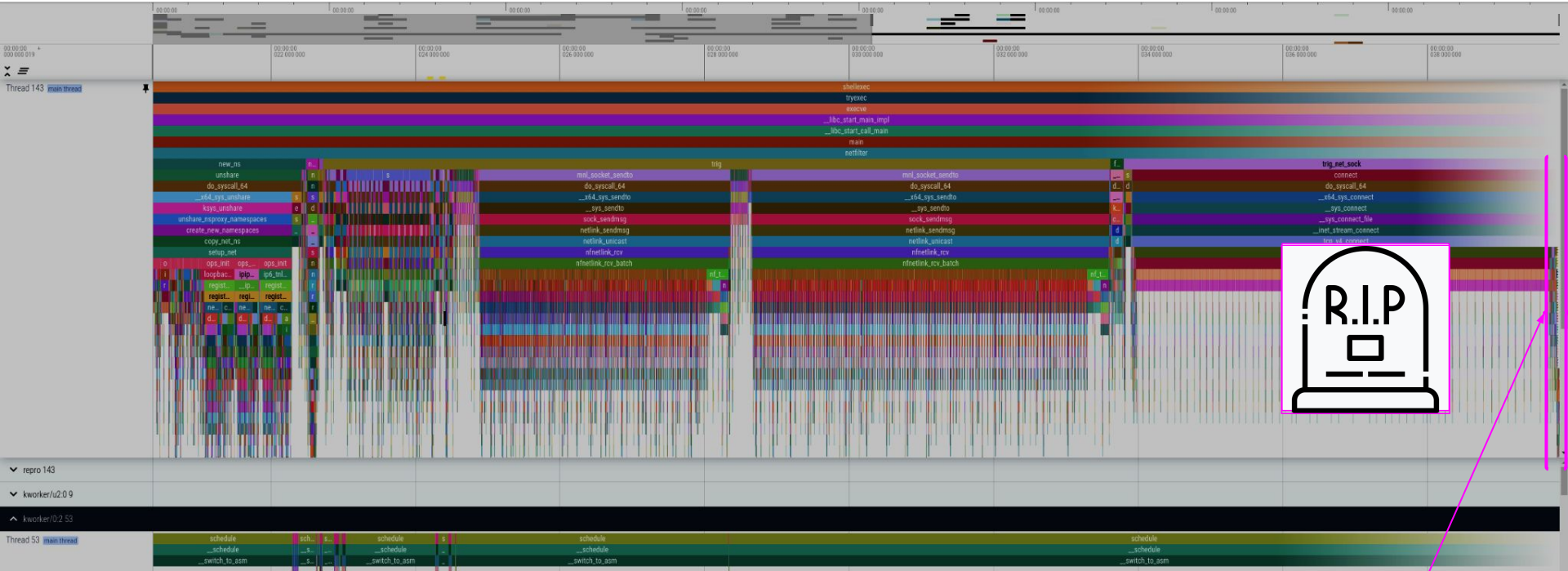
2. Trigger Free
4. Trigger UaF Read
7. Trigger UaF Read

1. New NS (to reach attack surface)
3. Spray object
5. Spray object
6. Spray object
8. Spray object



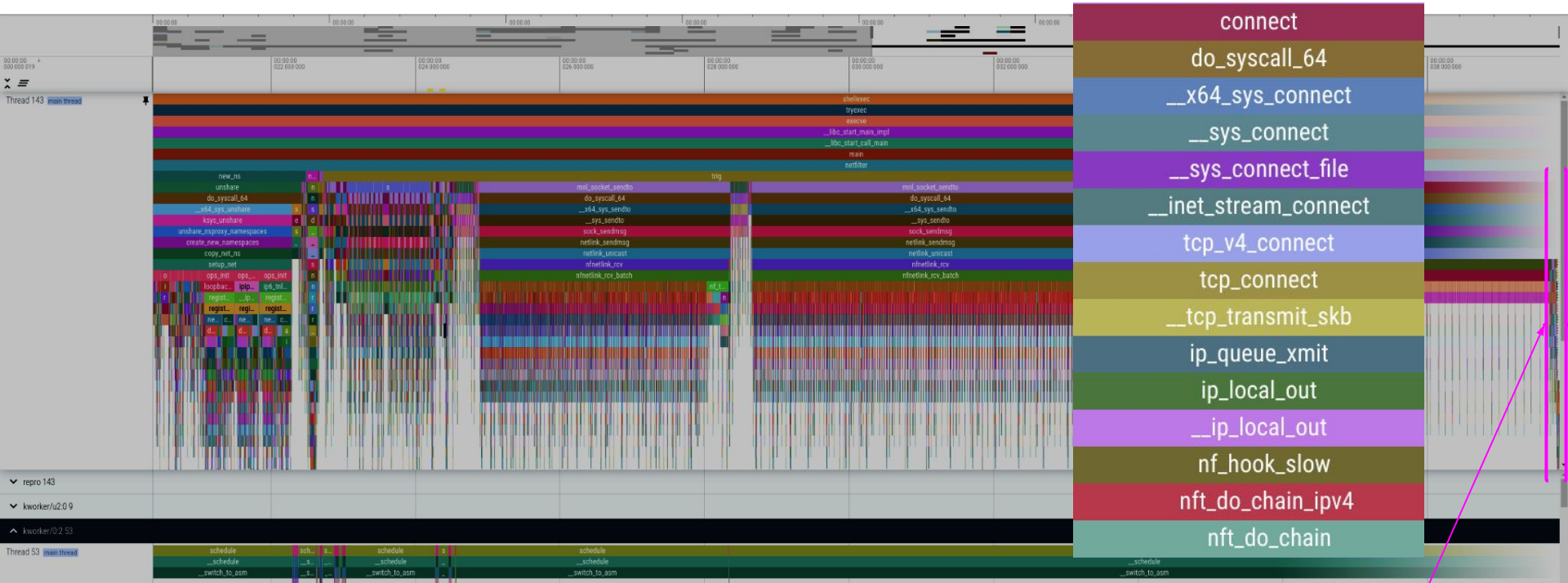
2. Trigger Free
4. Trigger UaF Read
7. Trigger UaF Read

1. New NS (to reach attack surface)
3. Spray object
5. Spray object
6. Spray object
8. Spray object



2. Trigger Free
4. Trigger UaF Read
7. Trigger UaF Read
9. Trigger UaF RIP control

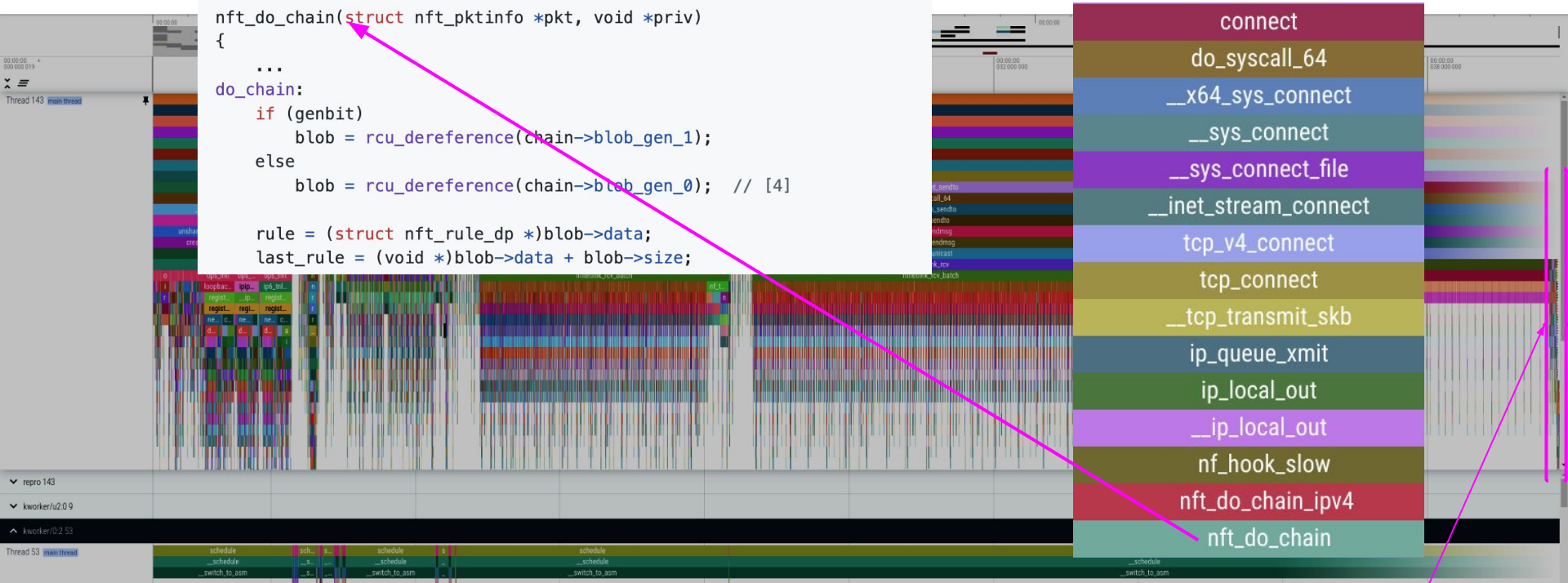
1. New NS (to reach attack surface)
3. Spray object
5. Spray object
6. Spray object
8. Spray object



2. Trigger Free
4. Trigger UaF Read
7. Trigger UaF Read
9. Trigger UaF RIP control

1. New NS (to reach attack surface)
3. Spray object
5. Spray object
6. Spray object
8. Spray object

```
nft_do_chain(struct nft_pktinfo *pkt, void *priv)
{
    ...
do_chain:
    if (genbit)
        blob = rcu_dereference(chain->blob_gen_1);
    else
        blob = rcu_dereference(chain->blob_gen_0);  // [4]

    rule = (struct nft_rule_dp *)blob->data;
    last_rule = (void *)blob->data + blob->size;
```

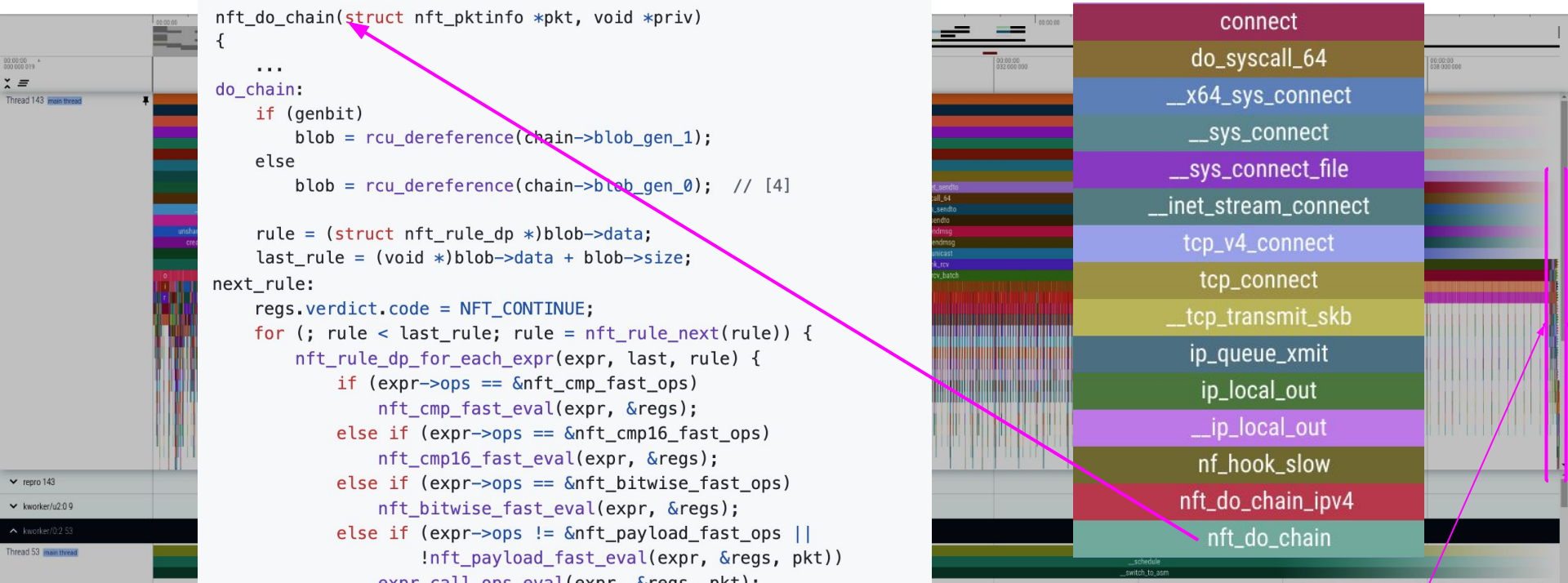


connect
do_syscall_64
__x64_sys_connect
__sys_connect
__sys_connect_file
__inet_stream_connect
tcp_v4_connect
tcp_connect
__tcp_transmit_skb
ip_queue_xmit
ip_local_out
__ip_local_out
nf_hook_slow
nft_do_chain_ipv4
nft_do_chain

2. Trigger Free
4. Trigger UaF Read
7. Trigger UaF Read
9. Trigger UaF RIP control

1. New NS (to reach attack surface)
3. Spray object
5. Spray object
6. Spray object
8. Spray object

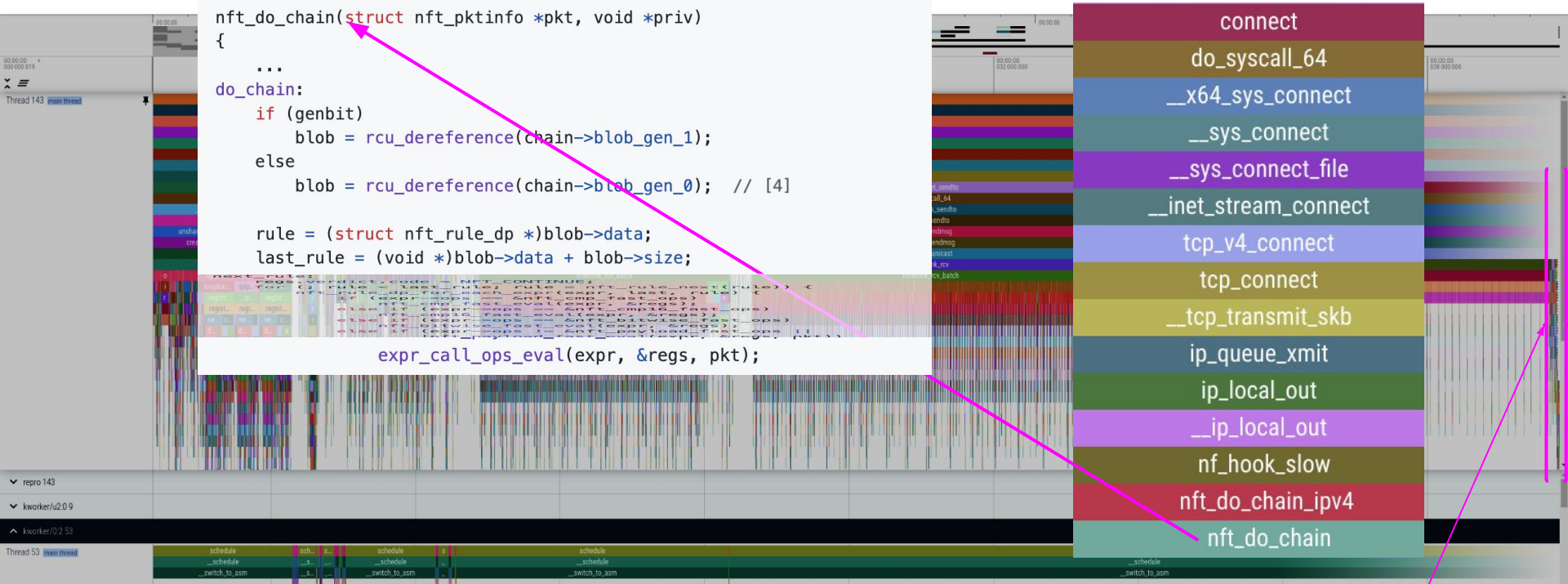
```
nft_do_chain(struct nft_pktinfo *pkt, void *priv)
{
    ...
do_chain:
    if (genbit)
        blob = rcu_dereference(chain->blob_gen_1);
    else
        blob = rcu_dereference(chain->blob_gen_0);  // [4]

    rule = (struct nft_rule_dp *)blob->data;
    last_rule = (void *)blob->data + blob->size;
next_rule:
    regs.verdict.code = NFT_CONTINUE;
    for (; rule < last_rule; rule = nft_rule_next(rule)) {
        nft_rule_dp_for_each_expr(expr, last, rule) {
            if (expr->ops == &nft_cmp_fast_ops)
                nft_cmp_fast_eval(expr, &regs);
            else if (expr->ops == &nft_cmp16_fast_ops)
                nft_cmp16_fast_eval(expr, &regs);
            else if (expr->ops == &nft_bitwise_fast_ops)
                nft_bitwise_fast_eval(expr, &regs);
            else if (expr->ops != &nft_payload_fast_ops ||
                    !nft_payload_fast_eval(expr, &regs, pkt))
                expr_call_ops_eval(expr, &regs, pkt);

            if (regs.verdict.code != NFT_CONTINUE)
                break;
        }
    ...
```



2. Trigger
er UaF

9. Trigger UaF RIP control

1. New NS (to reach attack surface)
3. Spray object
5. Spray object
6. Spray object
8. Spray object



2. Trigger Free
4. Trigger UaF Read
7. Trigger UaF Read
9. Trigger UaF RIP control

1. New NS (to reach attack surface)
3. Spray object
5. Spray object
6. Spray object
8. Spray object

```
nft_do_chain(struct nft_pktinfo *pkt, void *priv)
{
    ...
do_chain:
    if (genbit)
        blob = rcu_dereference(chain->blob_gen_1);
    else
        blob = rcu_dereference(chain->blob_gen_0);  // [4]

    rule = (struct nft_rule_dp *)blob->data;
    last_rule = (void *)blob->data + blob->size;
```

```
expr_call_ops_eval(expr, &regs, pkt);
```

```
static void expr_call_ops_eval(const struct nft_expr *expr,
                               struct nft_regs *regs,
                               struct nft_pktinfo *pkt)
{
#ifdef CONFIG_RETPOLINE
    unsigned long e = (unsigned long)expr->ops->eval;
#define X(e, fun) \
    do { if ((e) == (unsigned long)(fun)) \
        return fun(expr, regs, pkt); } while (0)  // [5]

    X(e, nft_payload_eval);
    X(e, nft_cmp_eval);
    X(e, nft_counter_eval);
```

- connect
- do_syscall_64
- __x64_sys_connect
- __sys_connect
- __sys_connect_file
- __inet_stream_connect
- tcp_v4_connect
- tcp_connect
- __tcp_transmit_skb
- ip_queue_xmit
- ip_local_out
- __ip_local_out
- nf_hook_slow
- nft_do_chain_ipv4
- nft_do_chain

2. Trigger … r UaF

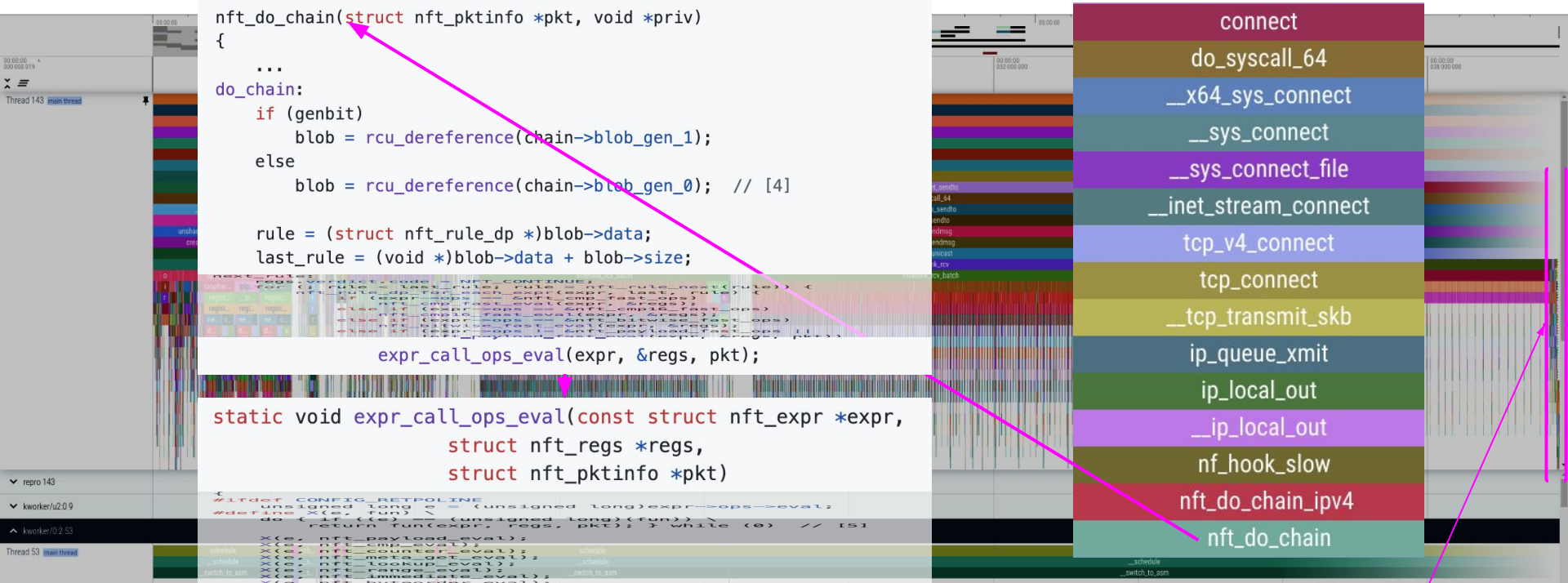
9. Trigger UaF RIP control

1. New NS (to reach attack surface)
3. Spray object
5. Spray object
6. Spray object
8. Spray object

```
nft_do_chain(struct nft_pktinfo *pkt, void *priv)
{
    ...
do_chain:
    if (genbit)
        blob = rcu_dereference(chain->blob_gen_1);
    else
        blob = rcu_dereference(chain->blob_gen_0);  // [4]

    rule = (struct nft_rule_dp *)blob->data;
    last_rule = (void *)blob->data + blob->size;
```

```
expr_call_ops_eval(expr, &regs, pkt);
```

```
static void expr_call_ops_eval(const struct nft_expr *expr,
                               struct nft_regs *regs,
                               struct nft_pktinfo *pkt)
{
#ifdef CONFIG_RETPOLINE
    unsigned long e = (unsigned long)expr->ops->eval;
#define X(e, fun) \
    do { if ((e) == (unsigned long)(fun)) \
        return fun(expr, regs, pkt); } while (0)   // [5]

    X(e, nft_payload_eval);
    X(e, nft_cmp_eval);
    X(e, nft_counter_eval);
    X(e, nft_meta_get_eval);
    X(e, nft_lookup_eval);
    X(e, nft_range_eval);
    X(e, nft_immediate_eval);
    X(e, nft_byteorder_eval);
    X(e, nft_dynset_eval);
    X(e, nft_rt_get_eval);
    X(e, nft_bitwise_eval);
#undef X
#endif /* CONFIG_RETPOLINE */

    expr->ops->eval(expr, regs, pkt);
}
```



2. Trigger
7. Trigger UaF

9. Trigger UaF RIP control

1. New NS (to reach attack surface)
3. Spray object
5. Spray object
6. Spray object
8. Spray object

```
nft_do_chain(struct nft_pktinfo *pkt, void *priv)
{
    ...
do_chain:
    if (genbit)
        blob = rcu_dereference(chain->blob_gen_1);
    else
        blob = rcu_dereference(chain->blob_gen_0);  // [4]

    rule = (struct nft_rule_dp *)blob->data;
    last_rule = (void *)blob->data + blob->size;
```

```
expr_call_ops_eval(expr, &regs, pkt);
```

```
static void expr_call_ops_eval(const struct nft_expr *expr,
                               struct nft_regs *regs,
                               struct nft_pktinfo *pkt)
```

```
    expr->ops->eval(expr, regs, pkt);
}
```

```
struct nft_chain {
        struct nft_rule_blob                    __rcu *blob_gen_0;
};
```

connect
do_syscall_64
__sys_connect_file
__inet_stream_connect
tcp_v4_connect
tcp_connect
__tcp_transmit_skb
ip_queue_xmit
ip_local_out
__ip_local_out
nf_hook_slow
nft_do_chain_ipv4
nft_do_chain

2. Trigger UaF
4. Trigger UaF
7. Trigger UaF
9. Trigger UaF RIP control

1. New NS (to reach attack surface)
3. Spray object
5. Spray object
6. Spray object
8. Spray object

```
nft_do_chain(struct nft_pktinfo *pkt, void *priv)
{
    ...
do_chain:
    if (genbit)
        blob = rcu_dereference(chain->blob_gen_1);
    else
        blob = rcu_dereference(chain->blob_gen_0);  // [4]

    rule = (struct nft_rule_dp *)blob->data;
    last_rule = (void *)blob->data + blob->size;
```

```
expr_call_ops_eval(expr, &regs, pkt);
```

```
static void expr_call_ops_eval(const struct nft_expr *expr,
                               struct nft_regs *regs,
                               struct nft_pktinfo *pkt)
{
#ifdef CONFIG_RETPOLINE
    unsigned long e = (unsigned long)expr->ops->eval;
#define X(e, fun) \
    do { if ((e) == (unsigned long)(fun)) \
        return fun(expr, regs, pkt); } while (0)   // [5]
    X(e, nft_payload_eval);
    X(e, nft_cmp_eval);
    X(e, nft_counter_eval);
    X(e, nft_meta_get_eval);
    X(e, nft_lookup_eval);
    X(e, nft_range_eval);
    X(e, nft_immediate_eval);
    X(e, nft_byteorder_eval);
    X(e, nft_dynset_eval);
    X(e, nft_rt_get_eval);
    X(e, nft_bitwise_eval);
#undef X
#endif /* CONFIG_RETPOLINE */
    expr->ops->eval(expr, regs, pkt);
}
```

```
struct nft_chain {
        struct nft_rule_blob                    __rcu *blob_gen_0;
};
```

```
struct nft_rule_dp {
        u64                                     is_last:1,
                                                dlen:12,
                                                handle:42;      /* for tracing */
        unsigned char                           data[]
                __attribute__((aligned(__alignof__(struct nft_expr))));
};
```

connect
do_syscall_64
__sys_connect_file
__tcp_transmit_skb
ip_queue_xmit
ip_local_out
__ip_local_out
nf_hook_slow
nft_do_chain_ipv4
nft_do_chain

2. Trigger UaF
7. Trigger UaF
9. Trigger UaF RIP control

1. New NS (to reach attack surface)
3. Spray object
5. Spray object
6. Spray object
8. Spray object

```
nft_do_chain(struct nft_pktinfo *pkt, void *priv)
{
    ...
do_chain:
    if (genbit)
        blob = rcu_dereference(chain->blob_gen_1);
    else
        blob = rcu_dereference(chain->blob_gen_0);  // [4]

    rule = (struct nft_rule_dp *)blob->data;
    last_rule = (void *)blob->data + blob->size;
```

```
expr_call_ops_eval(expr, &regs, pkt);
```

```
static void expr_call_ops_eval(const struct nft_expr *expr,
                               struct nft_regs *regs,
                               struct nft_pktinfo *pkt)
```

```
    expr->ops->eval(expr, regs, pkt);
}
```

```
struct nft_chain {
        struct nft_rule_blob                    __rcu *blob_gen_0;
};
```

```
struct nft_rule_dp {
        u64                                     is_last:1,
                                                dlen:12,
                                                handle:42;      /* for tracing */
        unsigned char                           data[]
                __attribute__((aligned(__alignof__(struct nft_expr))));
};
```

```
struct nft_expr {
        const struct nft_expr_ops               *ops;
        unsigned char                           data[]
                __attribute__((aligned(__alignof__(u64))));
};
```

connect
do_syscall_64
__sys_connect_file
__ip_local_out
nf_hook_slow
nft_do_chain_ipv4
nft_do_chain

2. Trigger
7. Trigger UaF

9. Trigger UaF RIP control

1. New NS (to reach attack surface)
3. Spray object
5. Spray object
6. Spray object
8. Spray object

```
nft_do_chain(struct nft_pktinfo *pkt, void *priv)
{
    ...
do_chain:
    if (genbit)
        blob = rcu_dereference(chain->blob_gen_1);
    else
        blob = rcu_dereference(chain->blob_gen_0);  // [4]

    rule = (struct nft_rule_dp *)blob->data;
    last_rule = (void *)blob->data + blob->size;
```

```
expr_call_ops_eval(expr, &regs, pkt);
```

```
static void expr_call_ops_eval(const struct nft_expr *expr,
                               struct nft_regs *regs,
                               struct nft_pktinfo *pkt)
{
    expr->ops->eval(expr, regs, pkt);
}
```

connect

do_syscall_64

```
struct nft_chain {
        struct nft_rule_blob            __rcu *blob_gen_0;
};
```

```
struct nft_rule_dp {
        u64                             is_last:1,
                                        dlen:12,
                                        handle:42;      /* for tracing */
        unsigned char                   data[]
                __attribute__((aligned(__alignof__(struct nft_expr))));
};
```

```
struct nft_expr {
        const struct nft_expr_ops       *ops;
        unsigned char                   data[]
                __attribute__((aligned(__alignof__(u64))));
};
```

```
struct nft_expr_ops {
        void    (*eval)(const struct nft_expr *expr,
                        struct nft_regs *regs,
                        const struct nft_pktinfo *pkt);
};
```

2. Trigger UaF
7. Trigger UaF
9. Trigger UaF RIP control

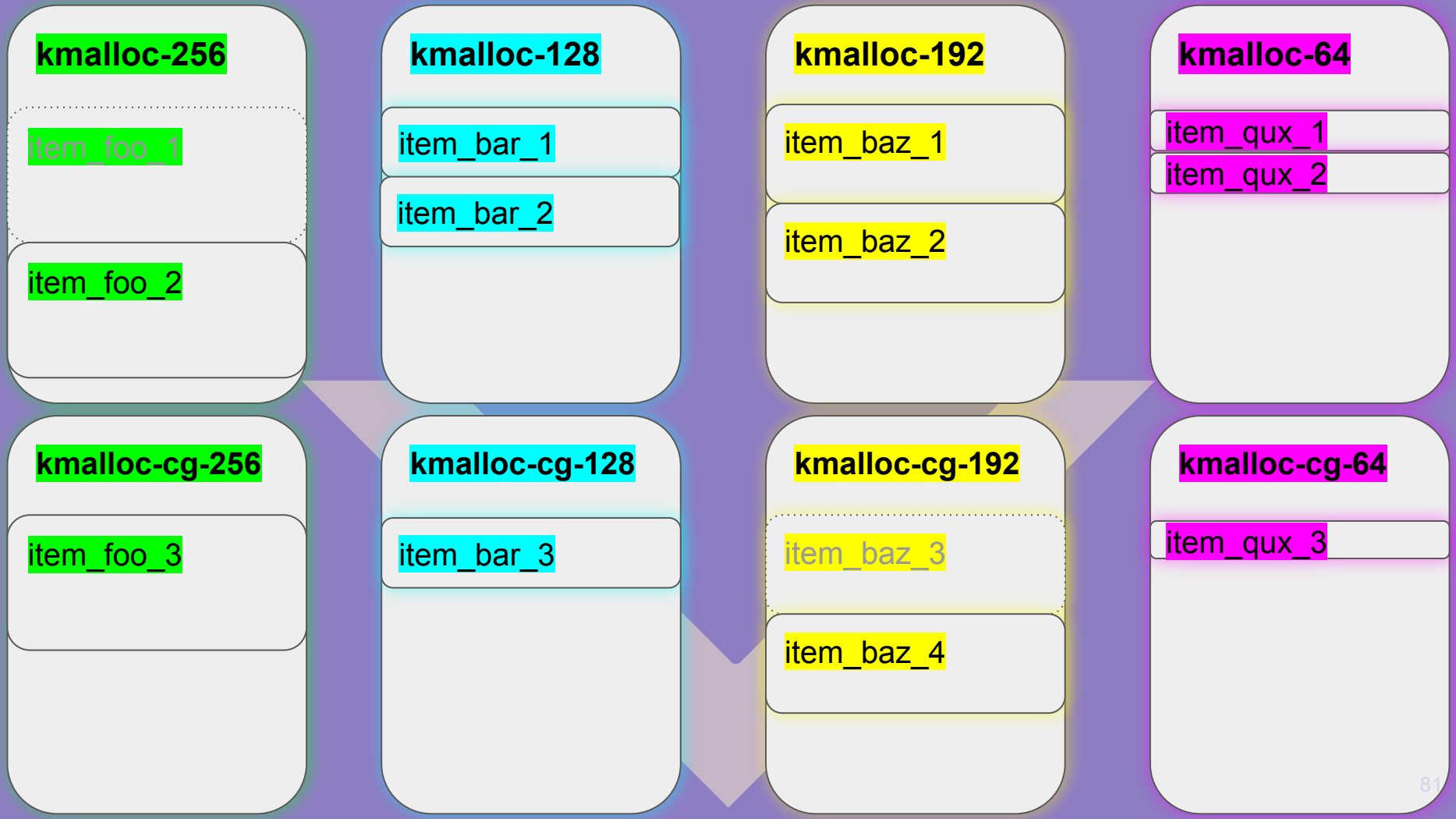
1. New NS (to reach attack surface)
3. Spray object
5. Spray object
6. Spray object
8. Spray object

```
struct nft_chain {
        struct nft_rule_blob            __rcu *blob_gen_0;
};
```

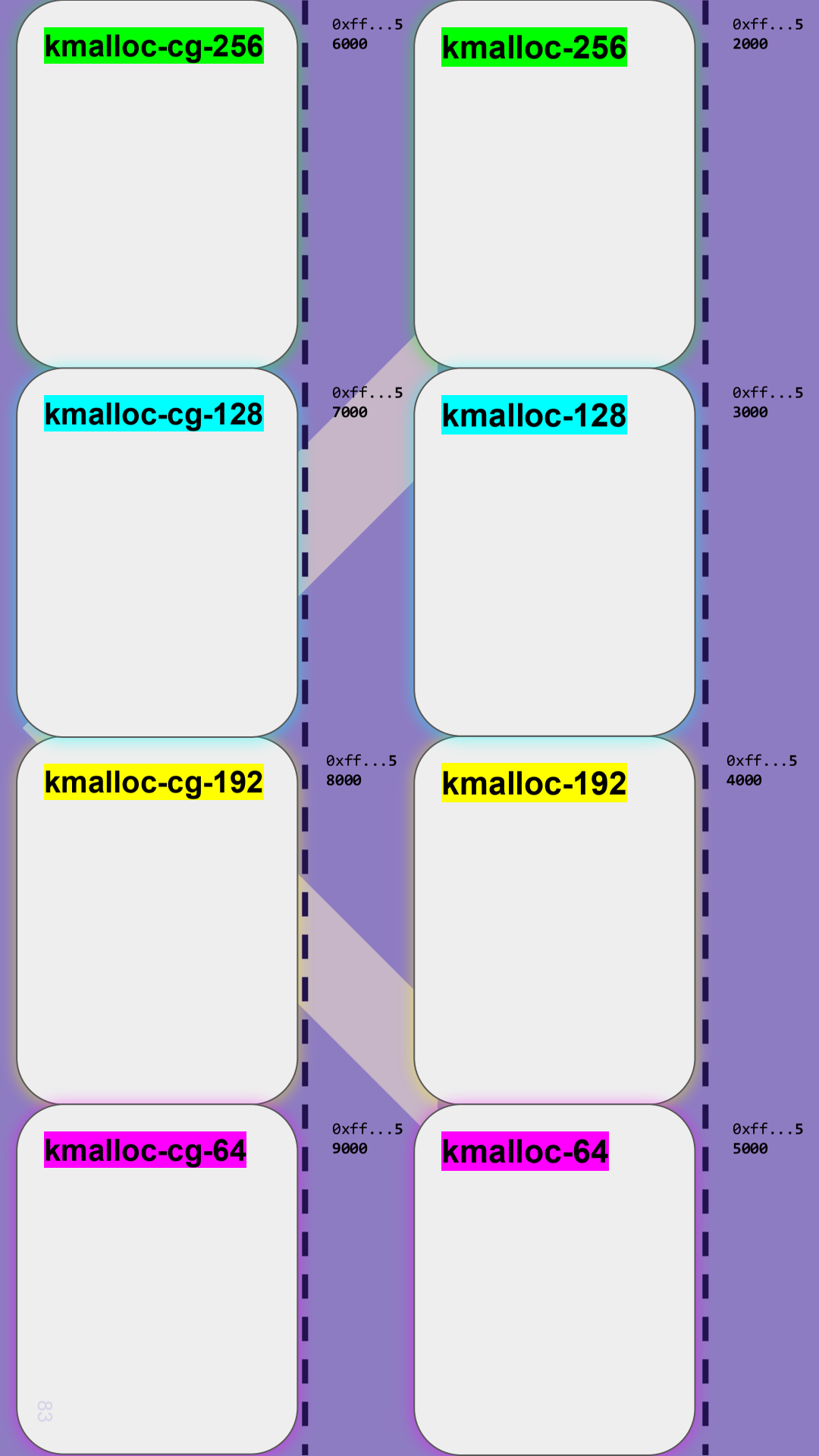
kmalloc-cg-64

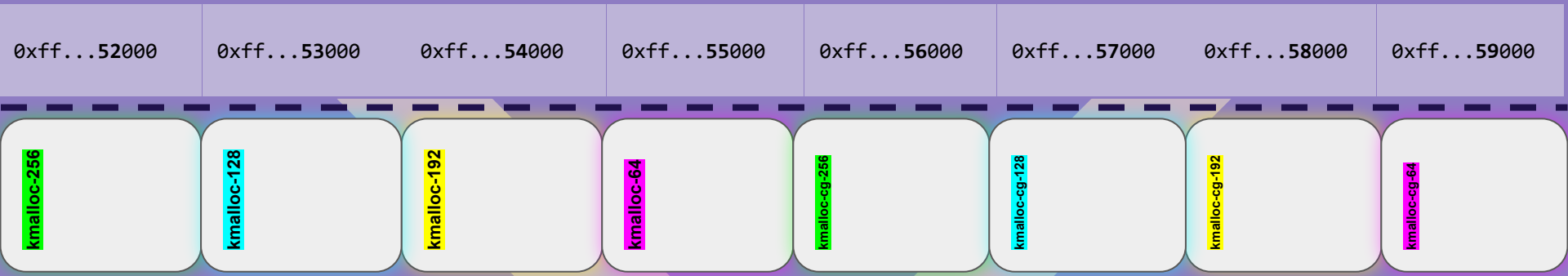
```
struct nft_rule_dp {
        u64                                     is_last:1,
                                                dlen:12,
                                                handle:42;      /* for tracing */

        struct nft_expr {
                const struct nft_expr_ops       *ops;
                unsigned char                   data[]
                        __attribute__((aligned(__alignof__(u64))));
        };
```

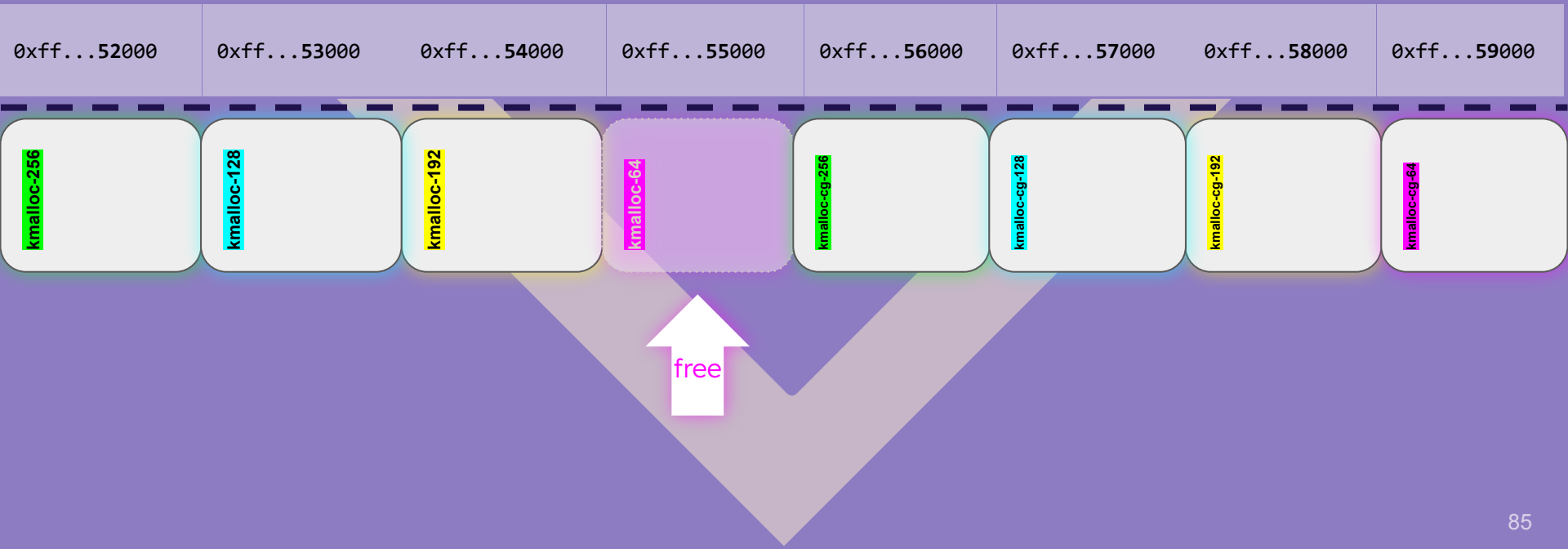
What if you can't spray on the correct slab?

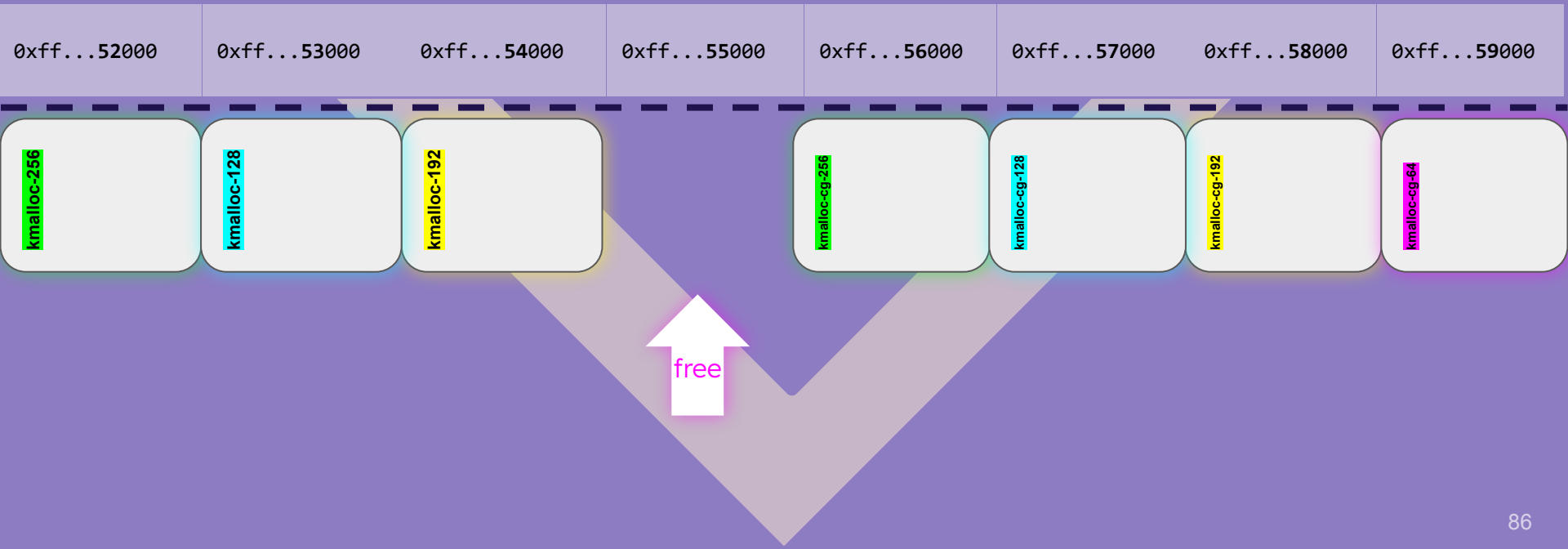

kmalloc-256
item_foo_1
item_foo_2
kmalloc-128
item_bar_1
item_bar_2
kmalloc-192
item_baz_1
item_baz_2
kmalloc-64
item_qux_1
item_qux_2
kmalloc-cg-256
item_foo_3
kmalloc-cg-128
item_bar_3
kmalloc-cg-192
item_baz_3
item_baz_4
kmalloc-cg-64
item_qux_3

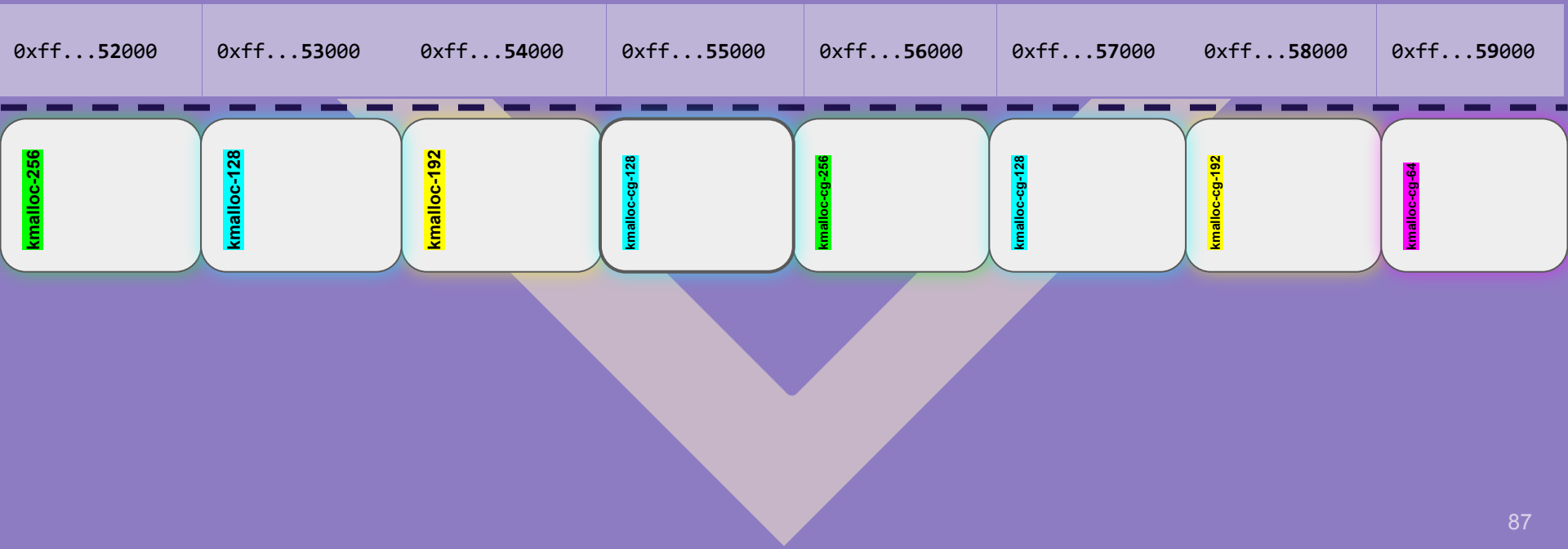

kmalloc-256
kmalloc-128
kmalloc-192
kmalloc-64
kmalloc-cg-256
kmalloc-cg-128
kmalloc-cg-192
kmalloc-cg-64

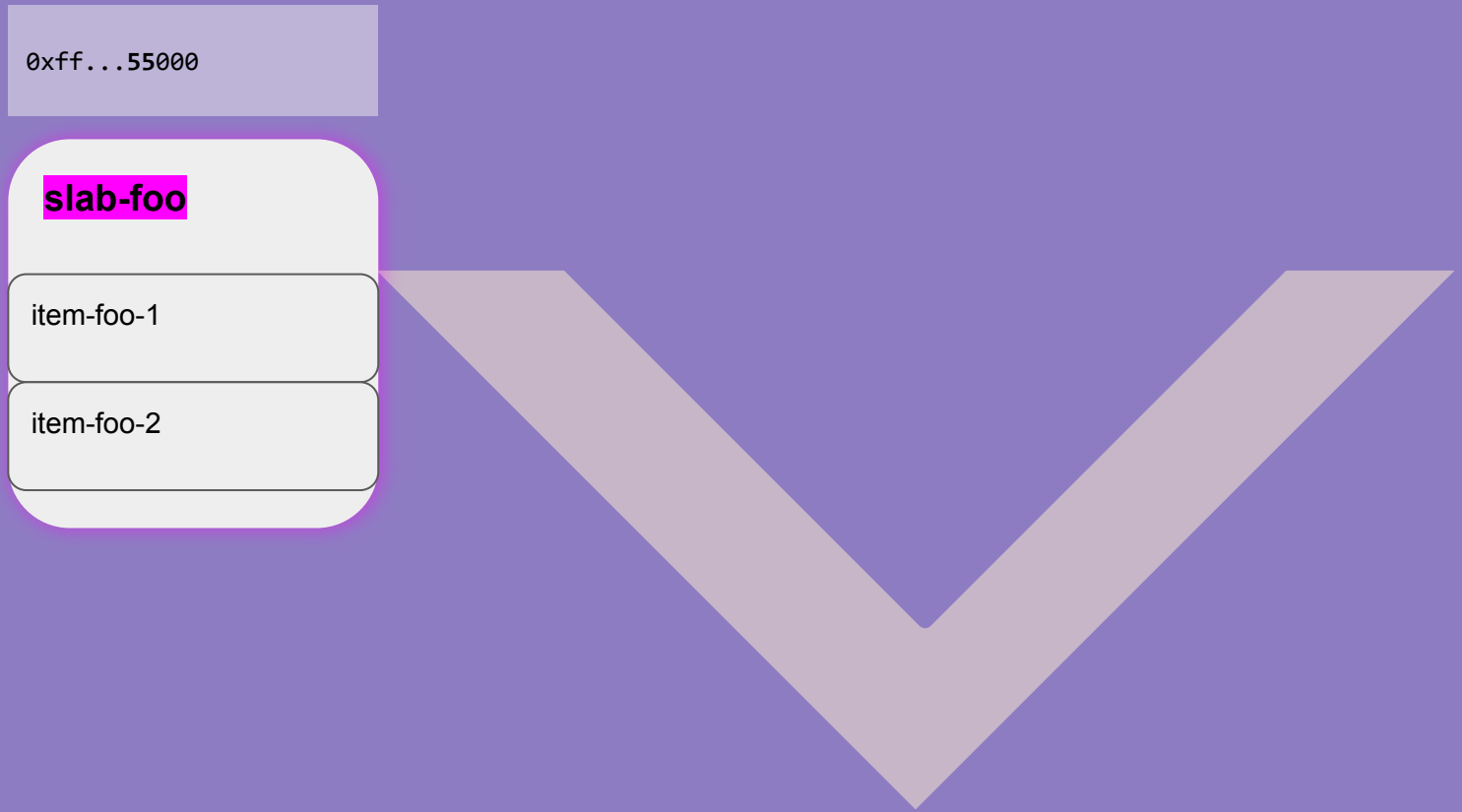

kmalloc-cg-256
0xff...5
6000
kmalloc-256
0xff...5
2000
kmalloc-cg-128
0xff...5
7000
kmalloc-128
0xff...5
3000
kmalloc-cg-192
0xff...5
8000
kmalloc-192
0xff...5
4000
kmalloc-cg-64
0xff...5
9000
kmalloc-64
0xff...5
5000

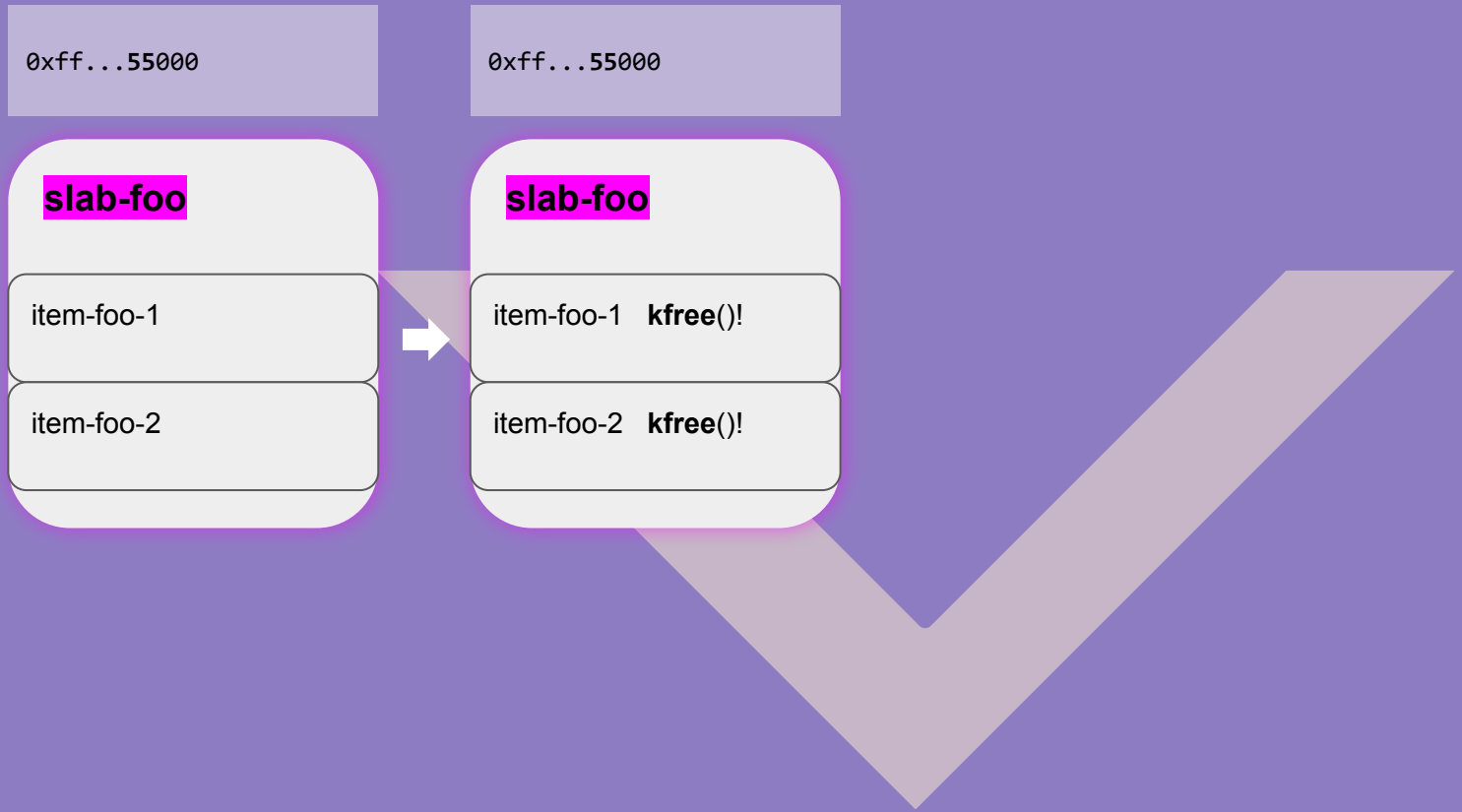

0xff...52000
0xff...53000
0xff...54000
0xff...55000
0xff...56000
0xff...57000
0xff...58000
0xff...59000
kmalloc-256
kmalloc-128
kmalloc-192
kmalloc-64
kmalloc-cg-256
kmalloc-cg-128
kmalloc-cg-192
kmalloc-cg-64

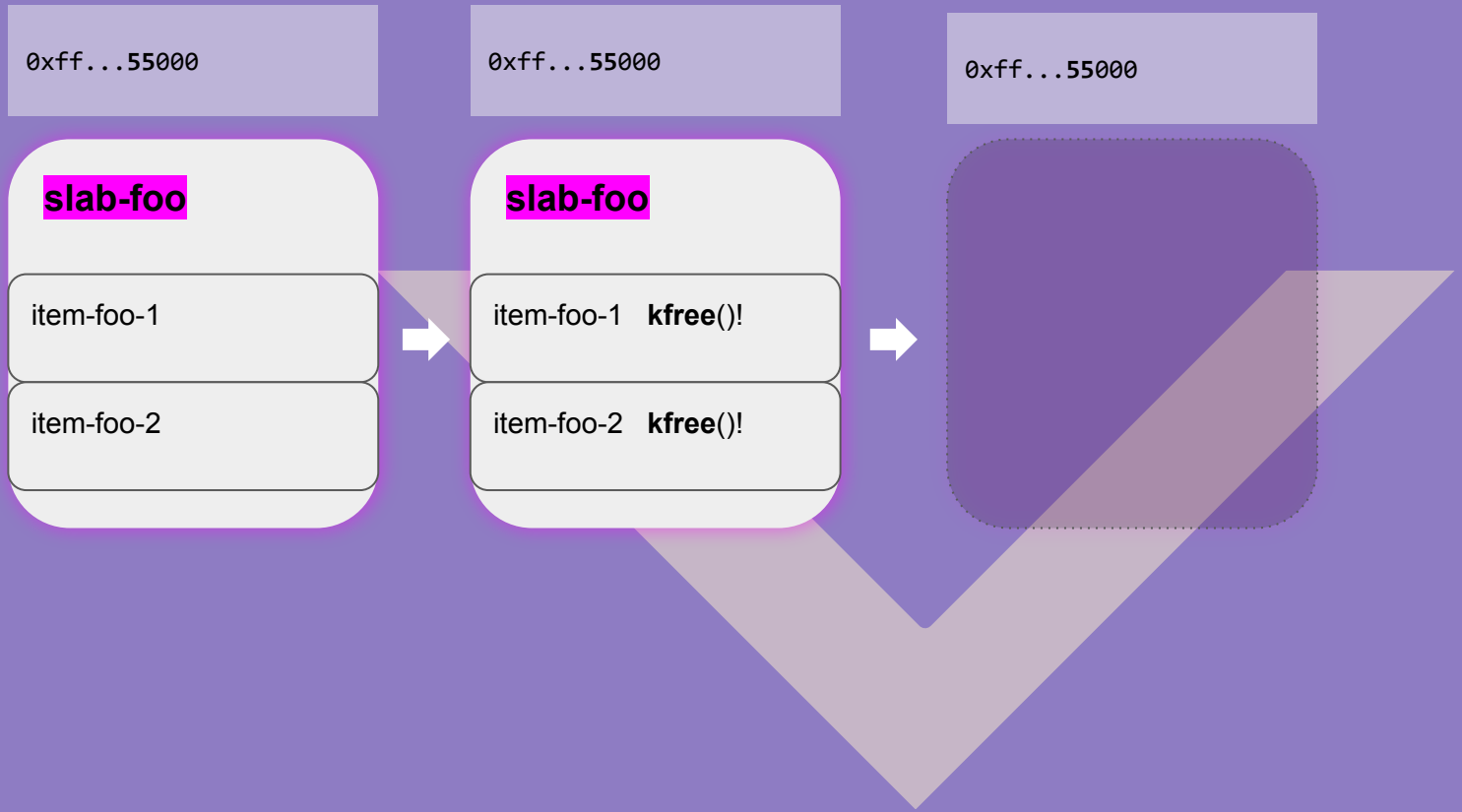

0xff...52000
0xff...53000
0xff...54000
0xff...55000
0xff...56000
0xff...57000
0xff...58000
0xff...59000
kmalloc-256
kmalloc-128
kmalloc-192
kmalloc-64
kmalloc-cg-256
kmalloc-cg-128
kmalloc-cg-192
kmalloc-cg-64
free

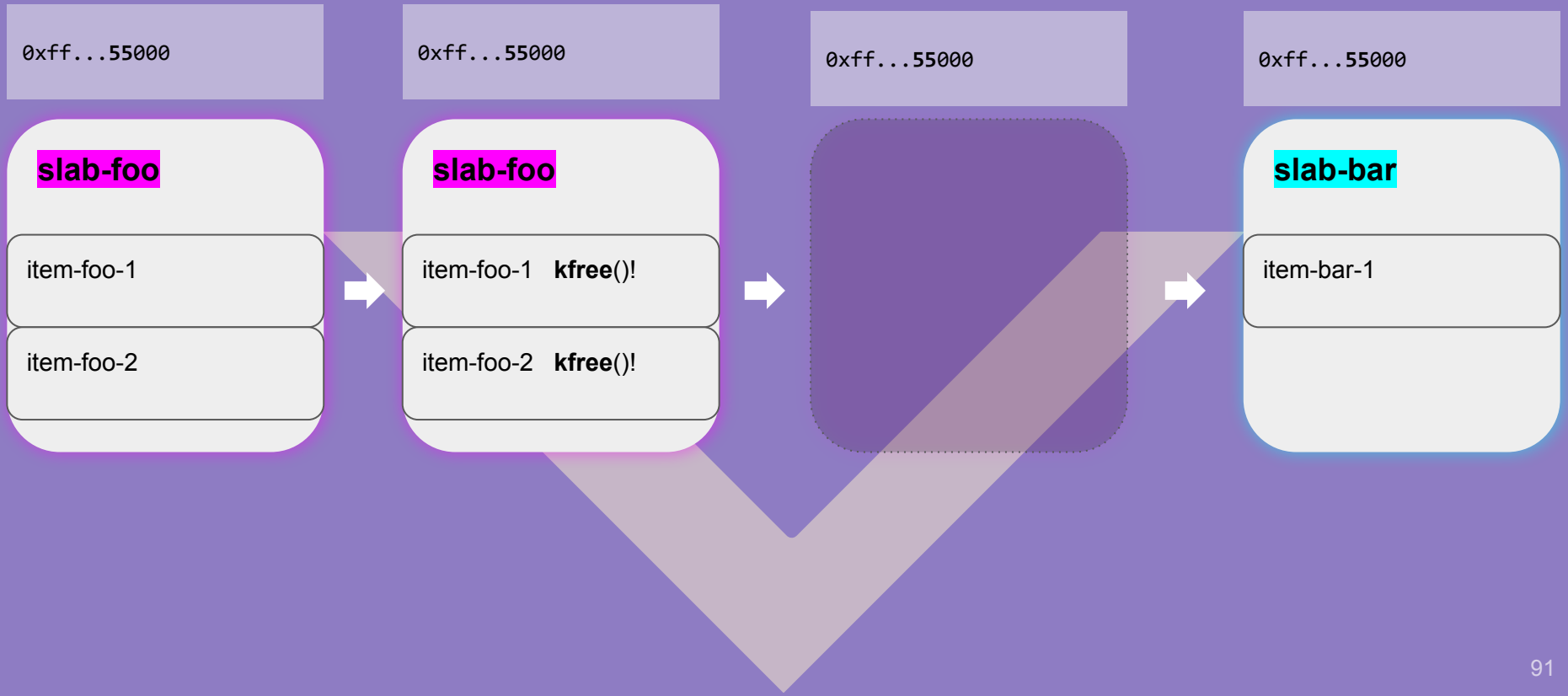

0xff...52000
0xff...53000
0xff...54000
0xff...55000
0xff...56000
0xff...57000
0xff...58000
0xff...59000
kmalloc-256
kmalloc-128
kmalloc-192
kmalloc-cg-256
kmalloc-cg-128
kmalloc-cg-192
kmalloc-cg-64
free


0xff...52000
0xff...53000
0xff...54000
0xff...55000
0xff...56000
0xff...57000
0xff...58000
0xff...59000
kmalloc-256
kmalloc-128
kmalloc-192
kmalloc-cg-128
kmalloc-cg-256
kmalloc-cg-128
kmalloc-cg-192
kmalloc-cg-64

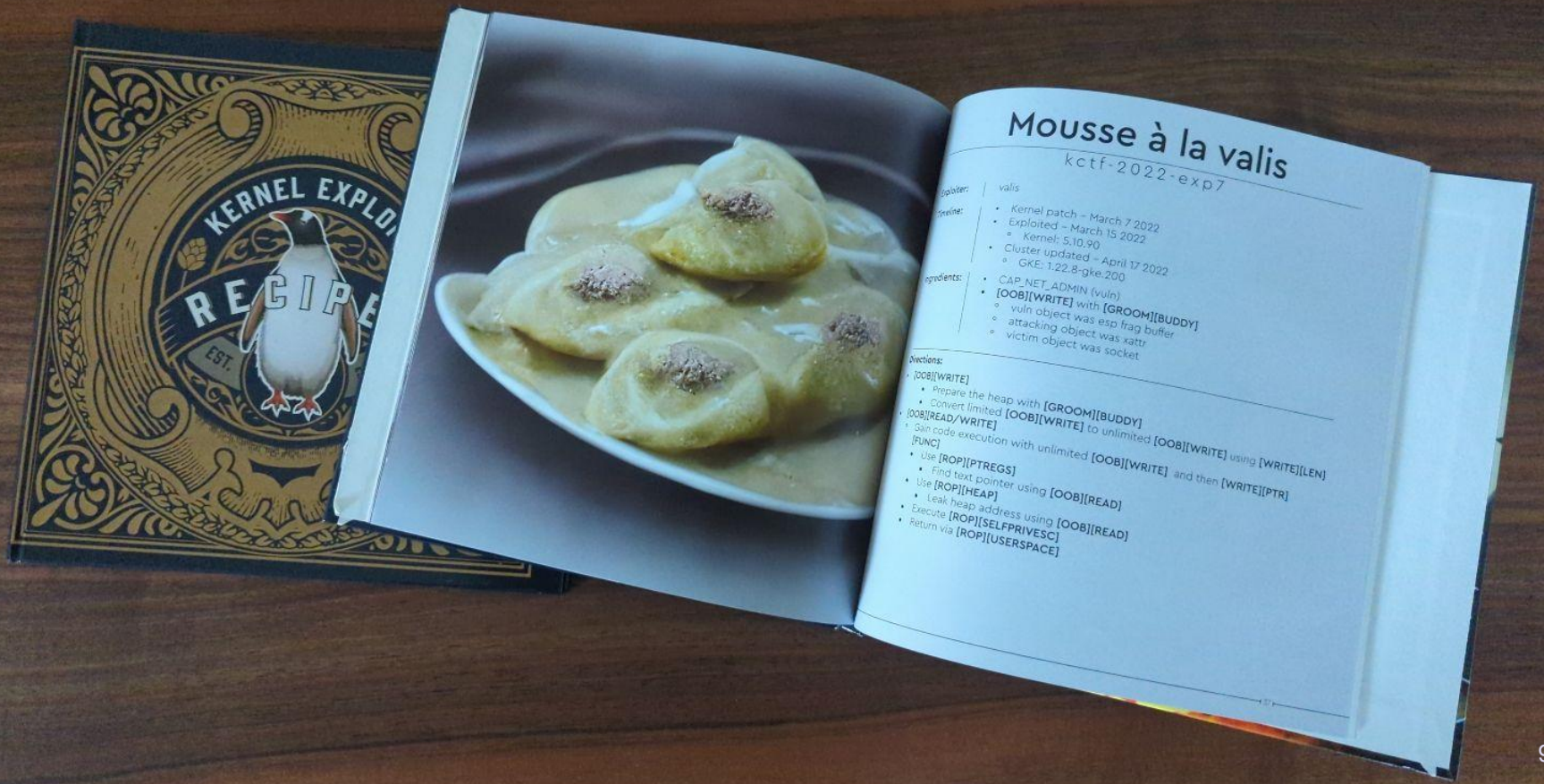

0xff...55000
slab-foo
item-foo-1
item-foo-2


0xff...55000
slab-foo
item-foo-1
item-foo-2
0xff...55000
slab-foo
item-foo-1 kfree()!
item-foo-2 kfree()!

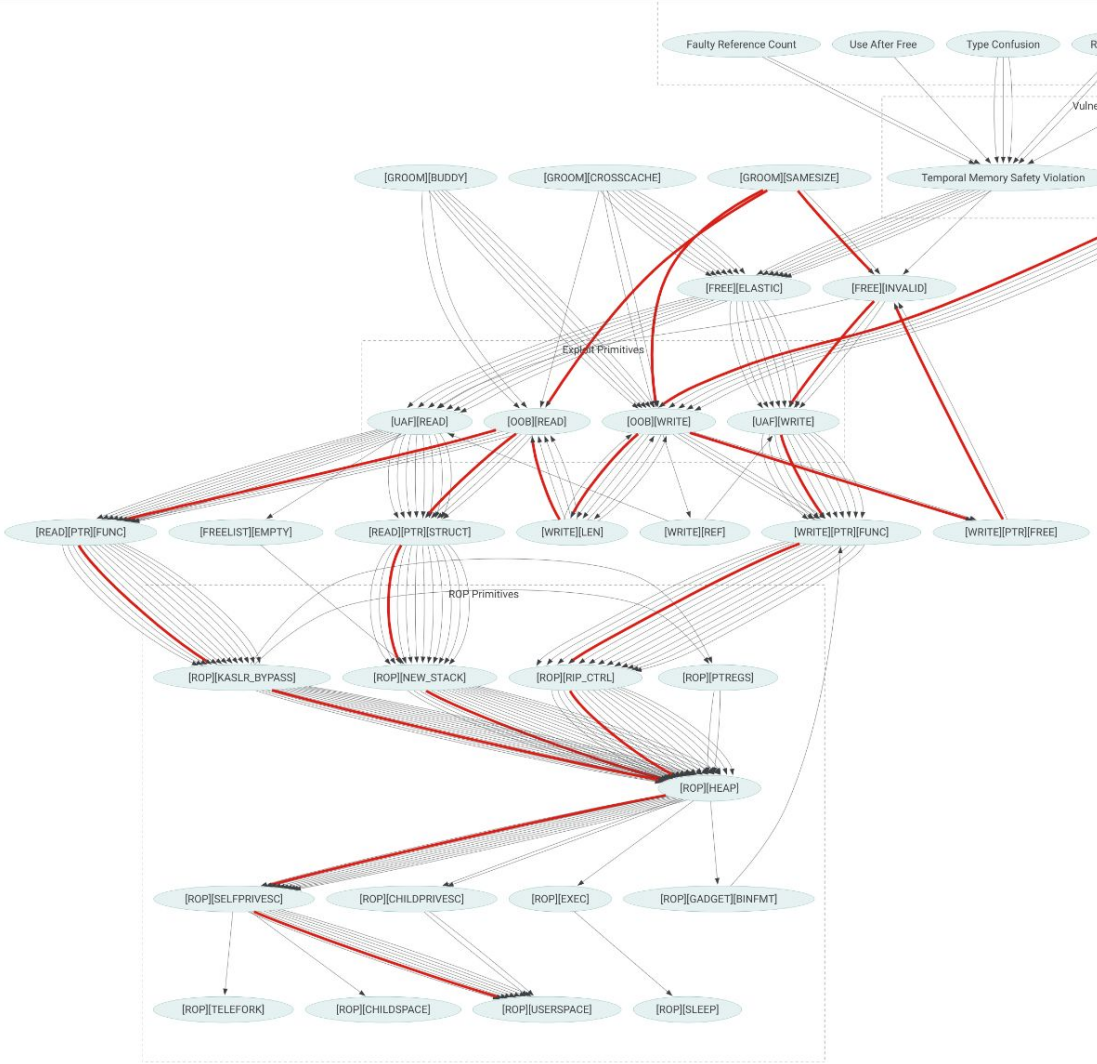

0xff...55000
slab-foo
item-foo-1
item-foo-2
0xff...55000
slab-foo
item-foo-1 kfree()!
item-foo-2 kfree()!
0xff...55000

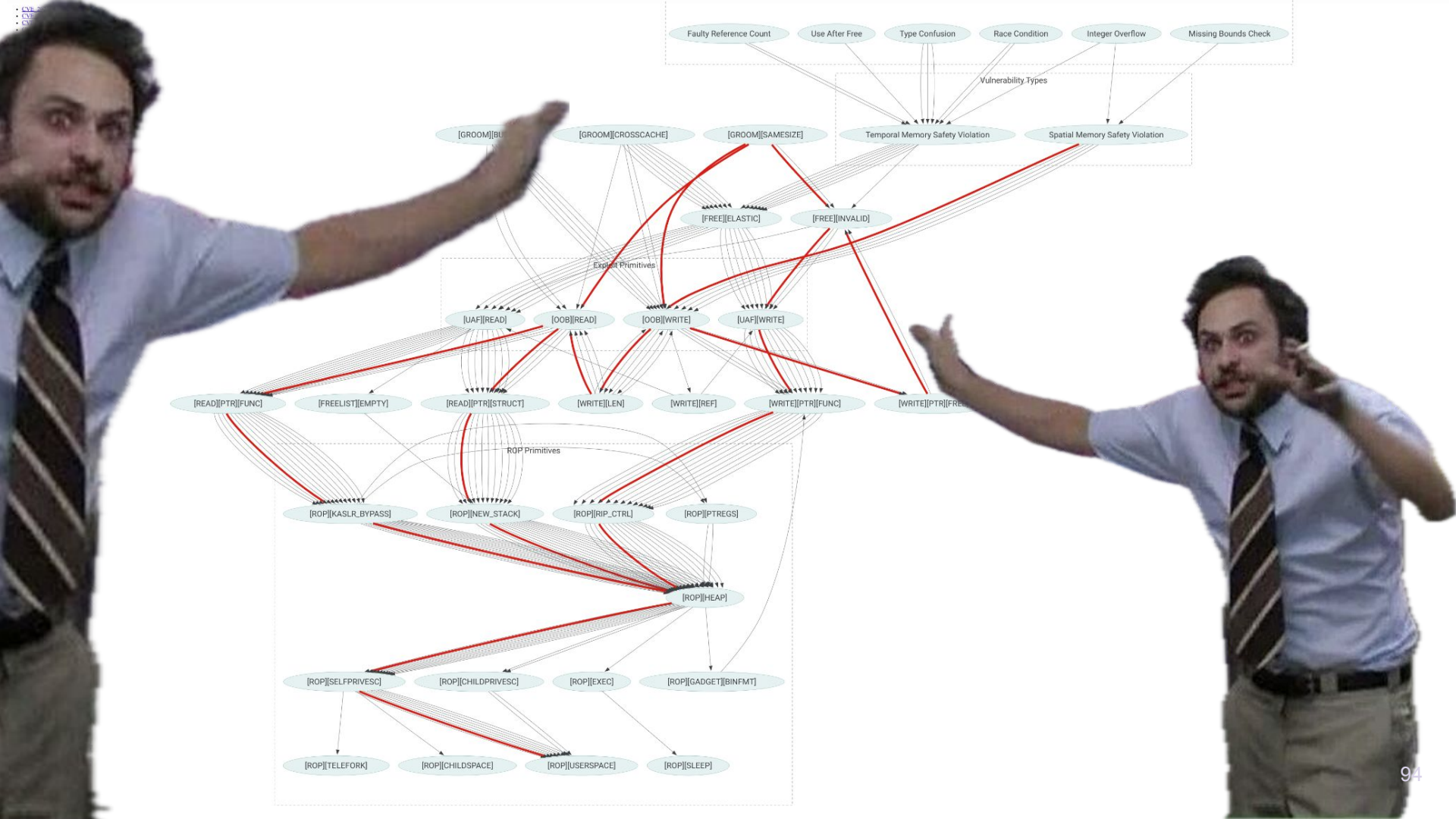

0xff...55000
slab-foo
item-foo-1
item-foo-2
0xff...55000
slab-foo
item-foo-1 kfree()!
item-foo-2 kfree()!
0xff...55000
0xff...55000
slab-bar
item-bar-1

Read online at
http://kernel.exploit.recipes

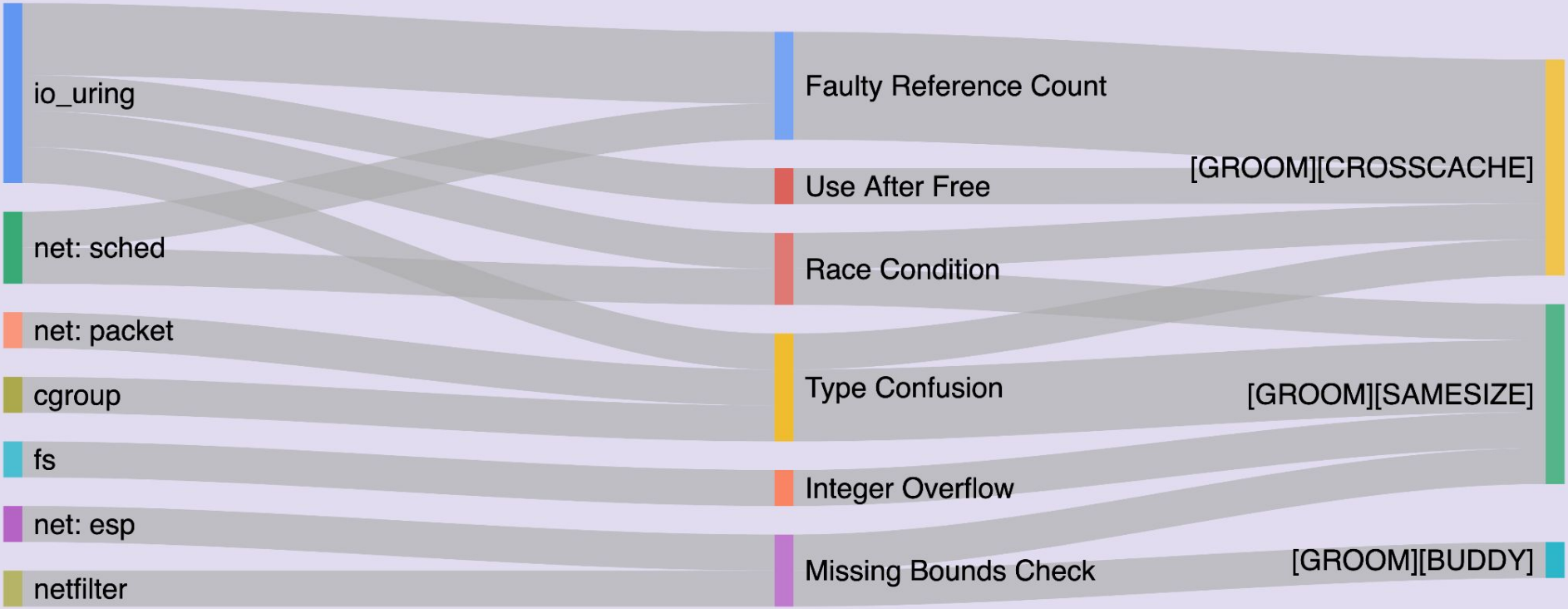


- CVE-2021-4154: exp1 exp2
- CVE-2021-22600: exp3
- CVE-2022-0185: exp4 exp5
- CVE-2022-27666: exp6 exp7 exp12
- CVE-2022-29582: exp9
- CVE-2022-1116: exp10
- CVE-2022-29581: exp11
- CVE-2022-1786: exp13
- CVE-2022-2327: exp14




Faulty Reference Count
Use After Free
Type Confusion
Race Condition
Integer Overflow
Missing Bounds Check
Vulnerability Types
[GROOM][CROSSCACHE]
[GROOM][SAMESIZE]
Temporal Memory Safety Violation
Spatial Memory Safety Violation
[FREE][ELASTIC]
[FREE][INVALID]
Exploit Primitives
[UAF][READ]
[OOB][READ]
[OOB][WRITE]
[UAF][WRITE]
[READ][PTR][FUNC]
[FREELIST][EMPTY]
[READ][PTR][STRUCT]
[WRITE][LEN]
[WRITE][REF]
[WRITE][PTR][FUNC]
ROP Primitives
[ROP][KASLR_BYPASS]
[ROP][NEW_STACK]
[ROP][RIP_CTRL]
[ROP][PTREGS]
[ROP][HEAP]
[ROP][SELFPRIVESC]
[ROP][CHILDPRIVESC]
[ROP][EXEC]
[ROP][GADGET][BINFMT]
[ROP][TELEFORK]
[ROP][CHILDSPACE]
[ROP][USERSPACE]
[ROP][SLEEP]

kCTF Exploits - ca. 2022



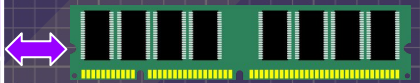
https://github.com/thejh/linux/blob/slub-virtual/MITIGATION_README

Jann Horn

1. Mitigates against cross-cache attacks
   - Virtual addresses for slabs are never reused
   - So you are forced to have objects on the same slab

```
reserve slab1
```

Virtual Address	Physical Address
0xFFFFFF0123456000	0x0000031234567890000



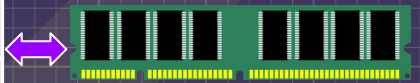
https://github.com/thejh/linux/blob/slub-virtual/MITIGATION_README

Jann Horn

1. Mitigates against cross-cache attacks
   - Virtual addresses for slabs are never reused
   - So you are forced to have objects on the same slab

`reserve slab1`

Virtual Address	Physical Address
0xFFFFFF0123456000	0x0000031234567890000



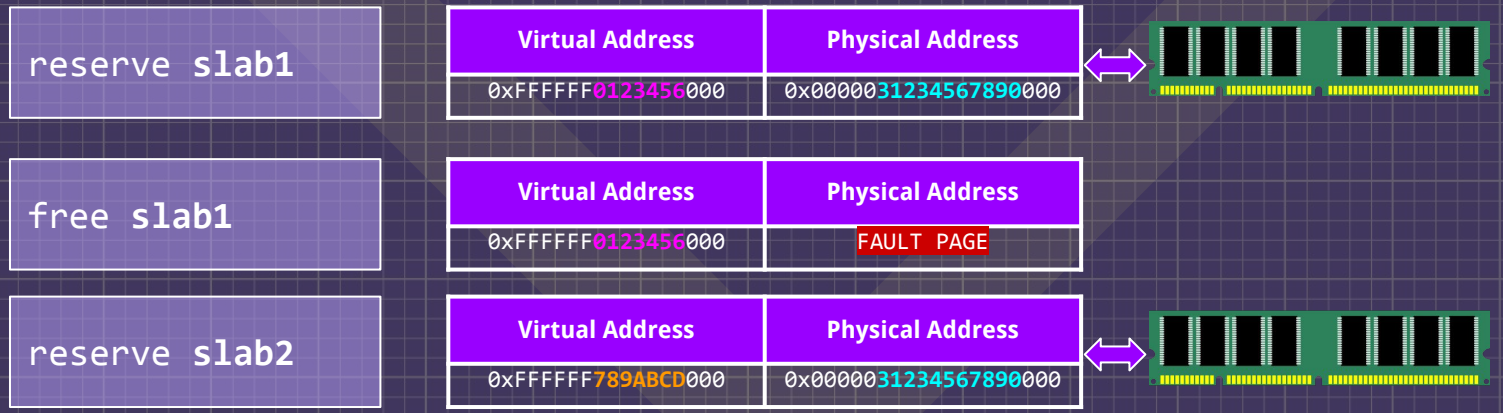
`free slab1`

Virtual Address	Physical Address
0xFFFFFF0123456000	FAULT PAGE

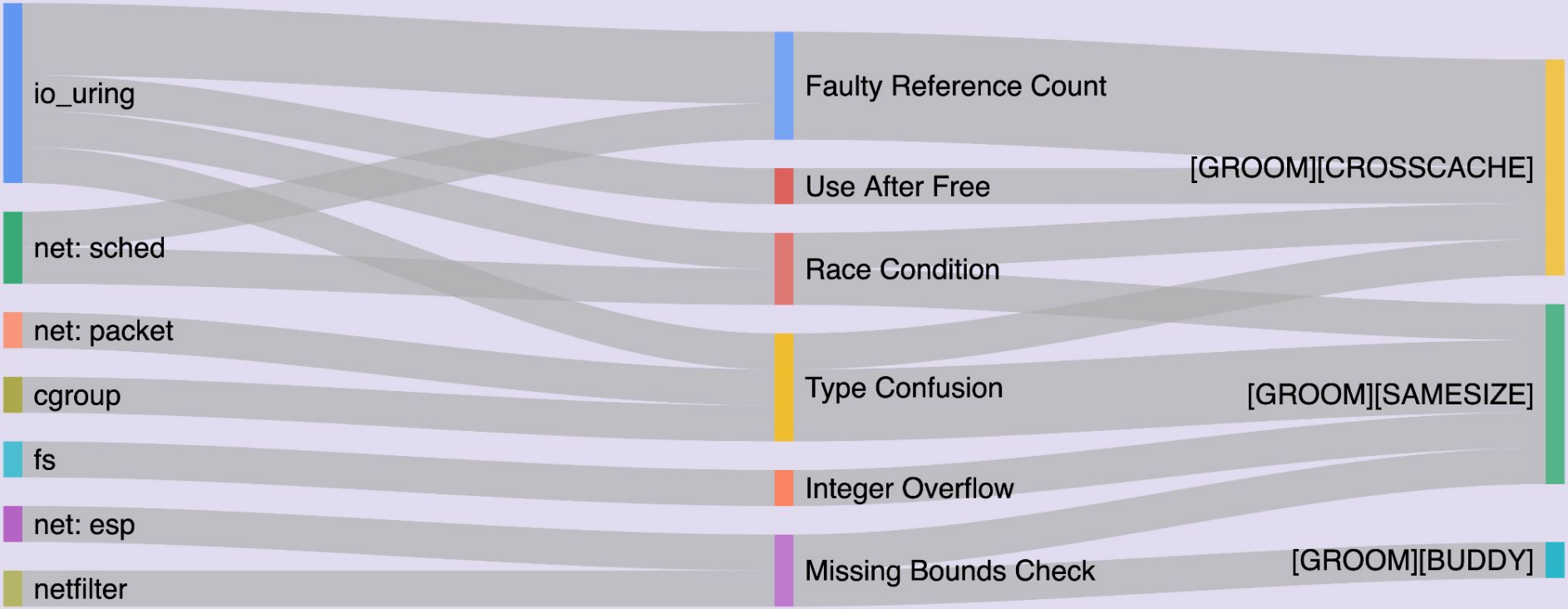
https://github.com/thejh/linux/blob/slub-virtual/MITIGATION_README

Jann Horn

1. Mitigates against cross-cache attacks
   - Virtual addresses for slabs are never reused
   - So you are forced to have objects on the same slab



kCTF Exploits - ca. 2022

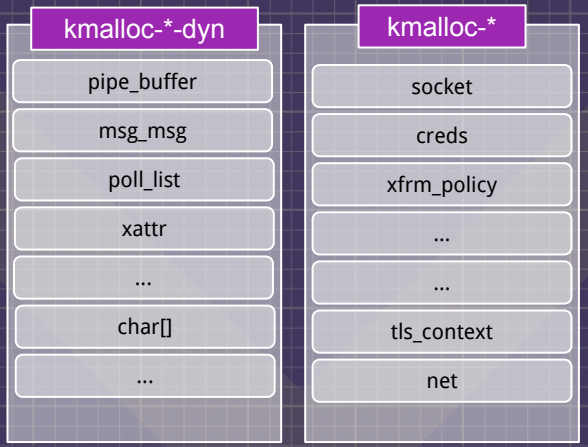


https://github.com/**thejh**/linux/blob/slub-virtual/MITIGATION_README

Jann Horn

2. Mitigates against variable size data structures

- Structs with variable size go on their own slab that isn't shared with provably-fixed-size objects
- Makes spraying a lot **harder**, but...



How useful is this mitigation?

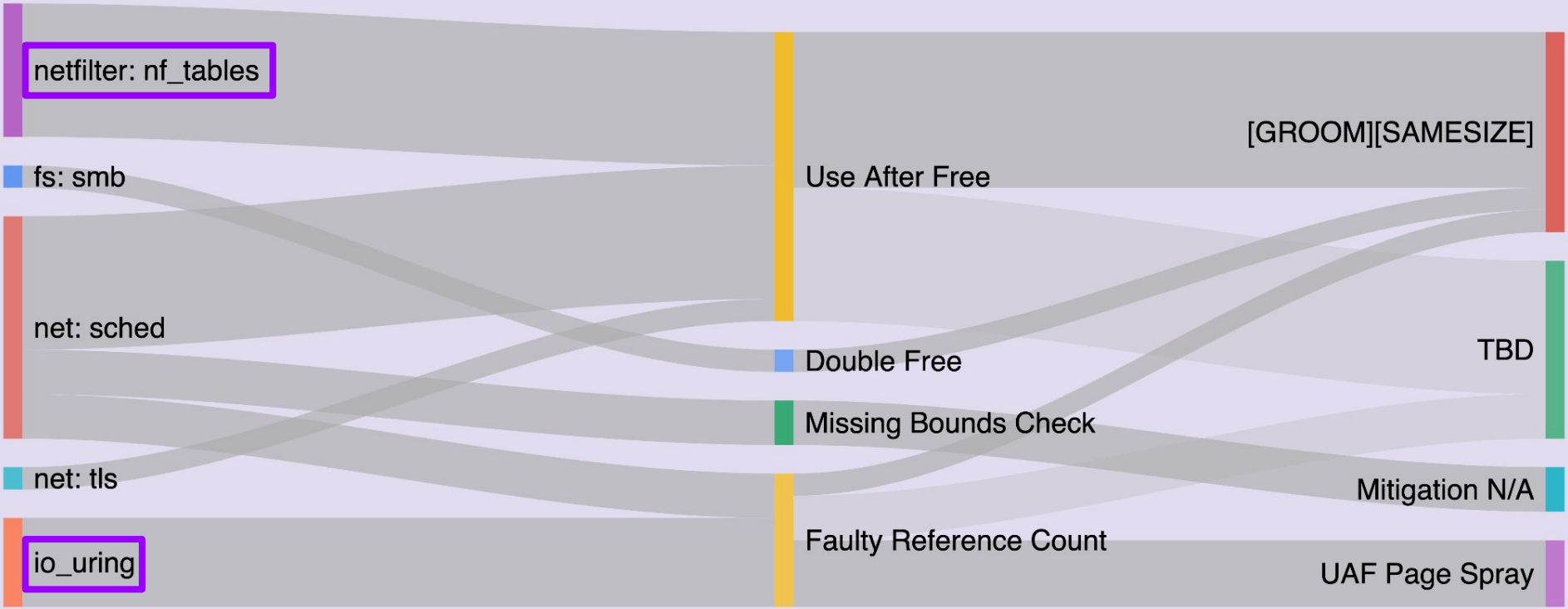
Google Security Blog

The latest news and insights from Google on security and safety on the Internet

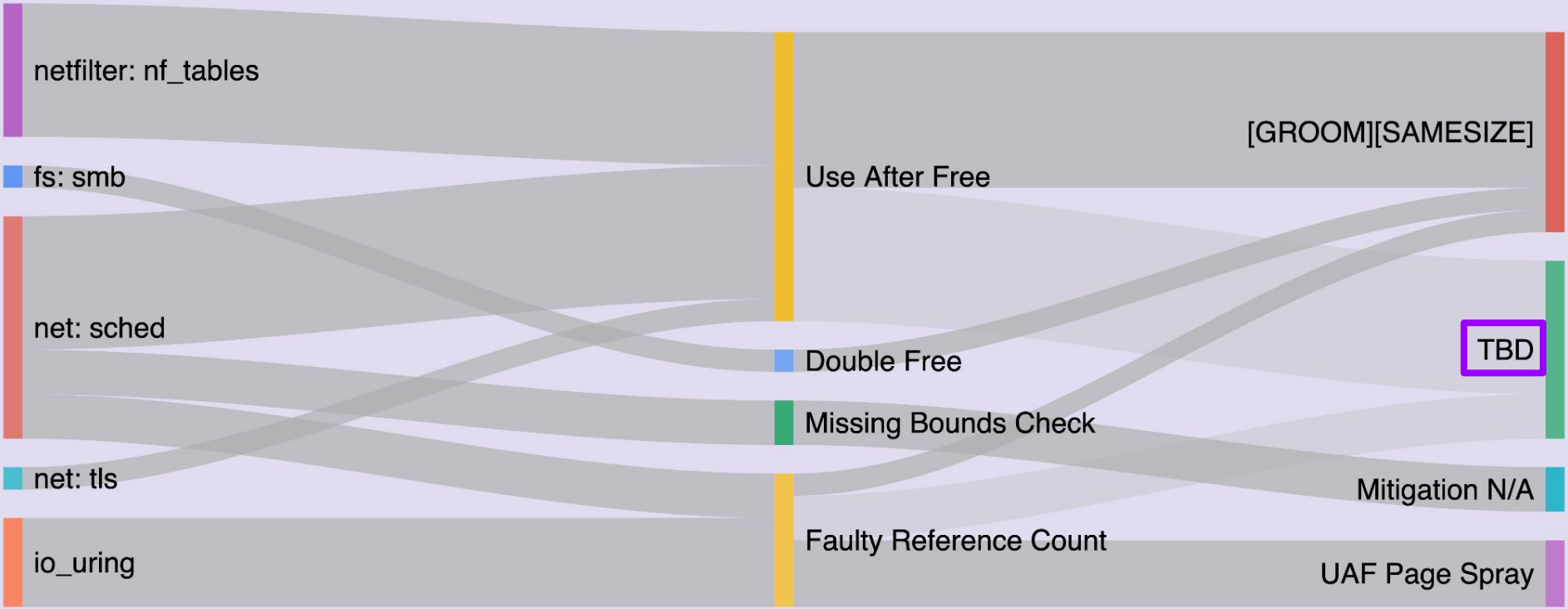
Making Linux Kernel Exploit Cooking Harder

August 10, 2022

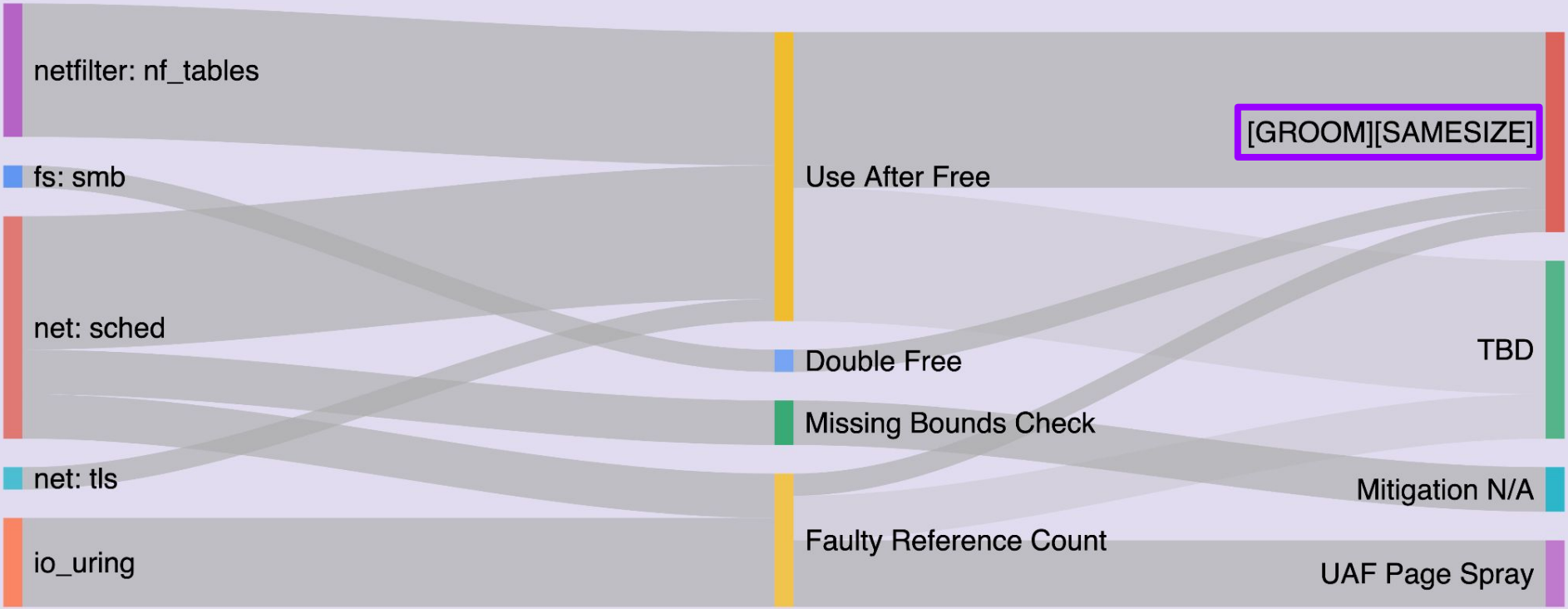
kCTF CVEs - Mitigation Instance - ca. 2023
SLAB_VIRTUAL + KMALLOC_SPLIT_VARSIZE



kCTF CVEs - Mitigation Instance - ca. 2023
SLAB_VIRTUAL + KMALLOC_SPLIT_VARSIZE



kCTF CVEs - Mitigation Instance - ca. 2023
SLAB_VIRTUAL + KMALLOC_SPLIT_VARSIZE



RANDOM_KMALLOC_CACHES

```
590  static __always_inline __alloc_size(1) void *kmalloc(size_t size, gfp_t flags)
591  {
592          if (__builtin_constant_p(size) && size) {
593                  unsigned int index;
594
595                  if (size > KMALLOC_MAX_CACHE_SIZE)
596                          return kmalloc_large(size, flags);
597
598                  index = kmalloc_index(size);
599                  return kmalloc_trace(
600                                  kmalloc_caches[kmalloc_type(flags, _RET_IP_)][index],
601                                  flags, size);
602          }
603          return __kmalloc(size, flags);
604  }
```

RANDOM_KMALLOC_CACHES

```
590  static __always_inline __alloc_size(1) void *kmalloc(size_t size, gfp_t flags)
591  {
592          if (__builtin_constant_p(size) && size) {
593                  unsigned int index;
594
595                  if (size > KMALLOC_MAX_CACHE_SIZE)
596                          return kmalloc_large(size, flags);
597
598                  index = kmalloc_index(size);
599                  return kmalloc_trace(
600                                  kmalloc_caches[kmalloc_type(flags, _RET_IP_)][index],
601                                  flags, size);
602
603
604
```

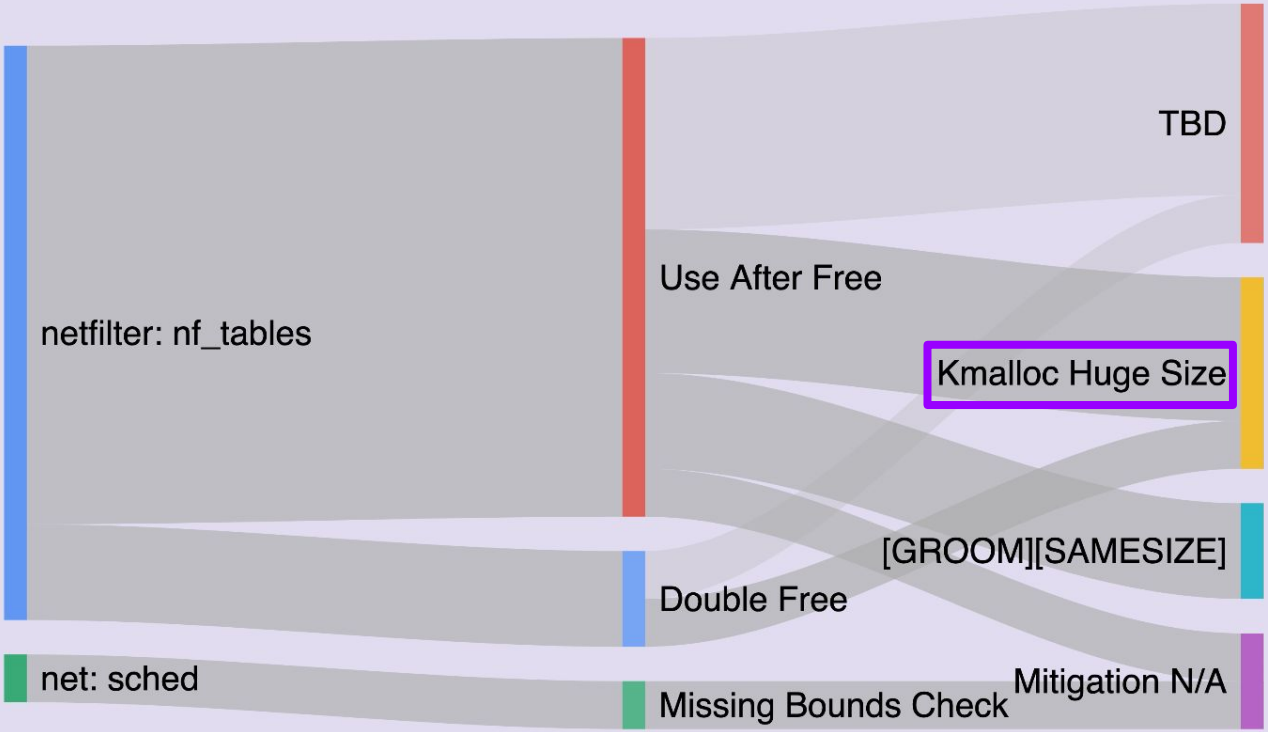
```
400  static __always_inline enum kmalloc_cache_type kmalloc_type(gfp_t flags, unsigned long caller)
401  {
402          /*
403           * The most common case is KMALLOC_NORMAL, so test for it
404           * with a single branch for all the relevant flags.
405           */
406          if (likely((flags & KMALLOC_NOT_NORMAL_BITS) == 0))
407  #ifdef CONFIG_RANDOM_KMALLOC_CACHES
408                  /* RANDOM_KMALLOC_CACHES_NR (=15) copies + the KMALLOC_NORMAL */
409                  return KMALLOC_RANDOM_START + hash_64(caller ^ random_kmalloc_seed,
410                                                        ilog2(RANDOM_KMALLOC_CACHES_NR + 1));
411  #else
```

RANDOM_KMALLOC_CACHES

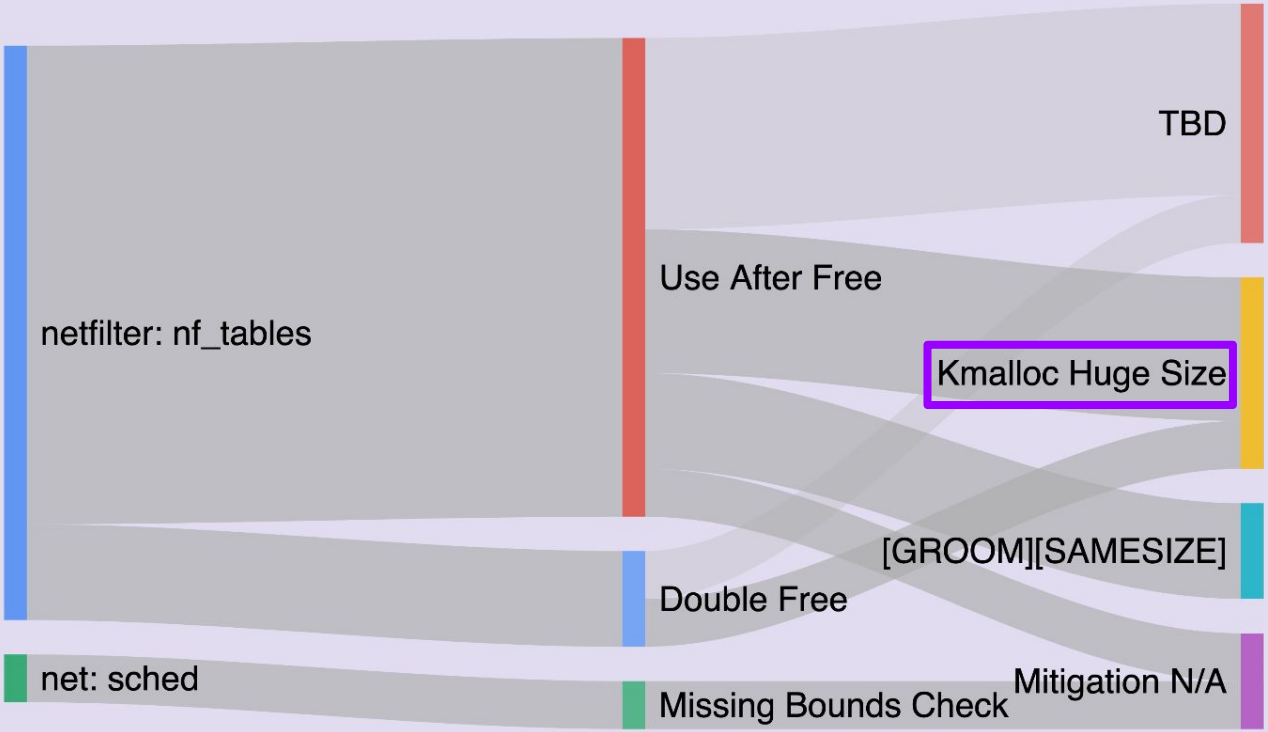

```
590  static __always_inline __alloc_size(1) void *kmalloc(size_t size, gfp_t flags)
591  {
592          if (__builtin_constant_p(size) && size) {
593                  unsigned int index;
594
595                  if (size > KMALLOC_MAX_CACHE_SIZE)
596                          return kmalloc_large(size, flags);
597
598                  index = kmalloc_index(size);
599                  return kmalloc_trace(
600                                  kmalloc_caches[kmalloc_type(flags, _RET_IP_)][index],
601                                  flags, size);
602
603
604
```

```
400  static __always_inline enum kmalloc_cache_type kmalloc_type(gfp_t flags, unsigned long caller)
401  {
402          /*
403           * The most common case is KMALLOC_NORMAL, so test for it
404           * with a single branch for all the relevant flags.
405           */
406          if (likely((flags & KMALLOC_NOT_NORMAL_BITS) == 0))
407  #ifdef CONFIG_RANDOM_KMALLOC_CACHES
408                  /* RANDOM_KMALLOC_CACHES_NR (=15) copies + the KMALLOC_NORMAL */
409                  return KMALLOC_RANDOM_START + hash_64(caller ^ random_kmalloc_seed,
410                                                        ilog2(RANDOM_KMALLOC_CACHES_NR + 1));
411  #else
```

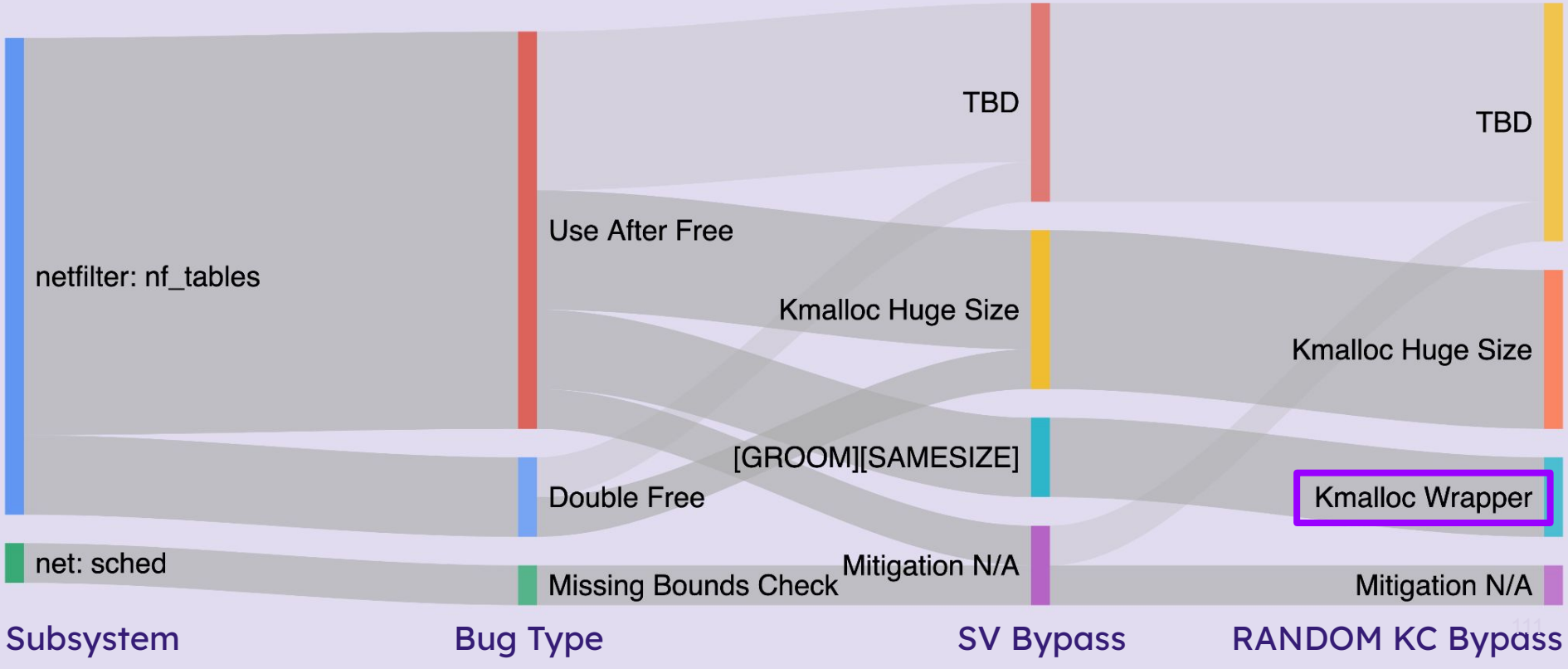
kCTF CVEs - Mitigation Instance - ca. 2024
… + RANDOM_KMALLOC_CACHES



kCTF CVEs - Mitigation Instance - ca
… + RANDOM_KMALLOC_CACHES



kCTF CVEs - Mitigation Instance - ca. 2024
… + RANDOM_KMALLOC_CACHES



liona24:CVE-2024-27397

```
static struct nft_rule_blob *nf_tables_chain_alloc_rules(unsigned int size)
{
        struct nft_rule_blob *blob;

        /* .. snip .. */

        blob = kvmalloc(size, GFP_KERNEL_ACCOUNT);

        /* .. snip .. */
        return blob;
}
```

liona24:CVE-2024-27397

```
static struct nft_rule_blob *nf_tables_chain_alloc_rules(unsigned int size)
{
        struct nft_rule_blob *blob;

        /* .. snip .. */

        blob = kvmalloc(size, GFP_KERNEL_ACCOUNT);

}
```

```
ffffffff813715f0 <kvmalloc_node>:
< .. snip .. >
ffffffff8137160e:       e9 dd cc 00 00          jmp    ffffffff8137e2f0 <__kmalloc_node>
ffffffff81371613:       89 f0                   mov    %esi,%eax
ffffffff81371615:       81 ce 00 20 01 00       or     $0x12000,%esi
ffffffff8137161b:       80 cc 20                or     $0x20,%ah
ffffffff8137161e:       f6 c7 40                test   $0x40,%bh
ffffffff81371621:       0f 45 f0                cmovne %eax,%esi
ffffffff81371624:       81 e6 ff 7f ff ff       and    $0xffff7fff,%esi
ffffffff8137162a:       e8 c1 cc 00 00          call   ffffffff8137e2f0 <__kmalloc_node>
< .. snip .. >
```

liona24:CVE-2024-27397

```
static struct nft_rule_blob *nf_tables_chain_alloc_rules(unsigned int size)
{
        struct nft_rule_blob *blob;

        /* .. snip .. */

        blob = kvmalloc(size, GFP_KERNEL_ACCOUNT);

}
```

```
ffffffff813715f0 <kvmalloc_node>:
< .. snip .. >
ffffffff8137160e:       e9 dd cc 00 00          jmp    ffffffff8137e2f0 <__kmalloc_node>
ffffffff81371613:       89 f0                   mov    %esi,%eax
ffffffff81371615:       81 ce 00 20 01 00       or     $0x12000,%esi
ffffffff8137161b:       80 cc 20                or     $0x20,%ah
ffffffff8137161e:       f6 c7 40                test   $0x40,%bh
ffffffff81371621:       0f 45 f0                cmovne %eax,%esi
ffffffff81371624:       81 e6 ff 7f ff ff       and    $0xffff7fff,%esi
ffffffff8137162a:       e8 c1 cc 00 00          call   ffffffff8137e2f0 <__kmalloc_node>
< .. snip .. >
```

liona24:CVE-2024-27397

```
static struct nft_rule_blob *nf_tables_chain_alloc_rules(unsigned int size)
{
        struct nft_rule_blob *blob;

        /* .. snip .. */

        blob = kvmalloc(size, GFP_KERNEL_ACCOUNT);
```

```
ffffffff813715f0 <kvmalloc_node>:
< .. snip .. >
ffffffff8137160e:       e9 dd cc 00 00          jmp    ffffffff8137e2f0 <__kmalloc_node>
ffffffff81371613:       89 f0                   mov    %esi,%eax
ffffffff81371615:       81 ce 00 20 01 00       or     $0x12000,%esi
ffffffff8137161b:       80 cc 20                or     $0x20,%ah
ffffffff8137161e:       f6 c7 40                test   $0x40,%bh
ffffffff81371621:       0f 45 f0                cmovne %eax,%esi
ffffffff81371624:       81 e6 ff 7f ff ff       and    $0xffff7fff,%esi
ffffffff8137162a:       e8 c1 cc 00 00          call   ffffffff8137e2f0 <__kmalloc_node>
< .. snip .. >
```

liona24:CVE-2024-27397

```
static struct nft_rule_blob *nf_tables_chain_alloc_rules(unsigned int size)
{
        struct nft_rule_blob *blob;

        /* .. snip .. */

        blob = kvmalloc(size, GFP_KERNEL_ACCOUNT);

}
```

```
ffffffff813715f0 <kvmalloc_node>:
< .. snip .. >
ffffffff8137160e:       e9 dd cc 00 00          jmp    ffffffff8137e2f0 <__kmalloc_node>
ffffffff81371613:       89 f0                   mov    %esi,%eax
ffffffff81371615:       81 ce 00 20 01 00       or     $0x12000,%esi
ffffffff8137161b:       80 cc 20                or     $0x20,%ah
ffffffff8137161e:       f6 c7 40                test   $0x40,%bh
ffffffff81371621:       0f 45 f0                cmovne %eax,%esi
ffffffff81371624:       81 e6 ff 7f ff ff       and    $0xffff7fff,%esi
ffffffff8137162a:       e8 c1 cc 00 00          call   ffffffff8137e2f0 <__kmalloc_node>
< .. snip .. >
```

liona24:CVE-2024-27397


```
static struct nft_rule_blob *nf_tables_chain_alloc_rules(unsigned int size)
{
        struct nft_rule_blob *blob;

        /* .. snip .. */

        blob = kvmalloc(size, GFP_KERNEL_ACCOUNT);

}
```

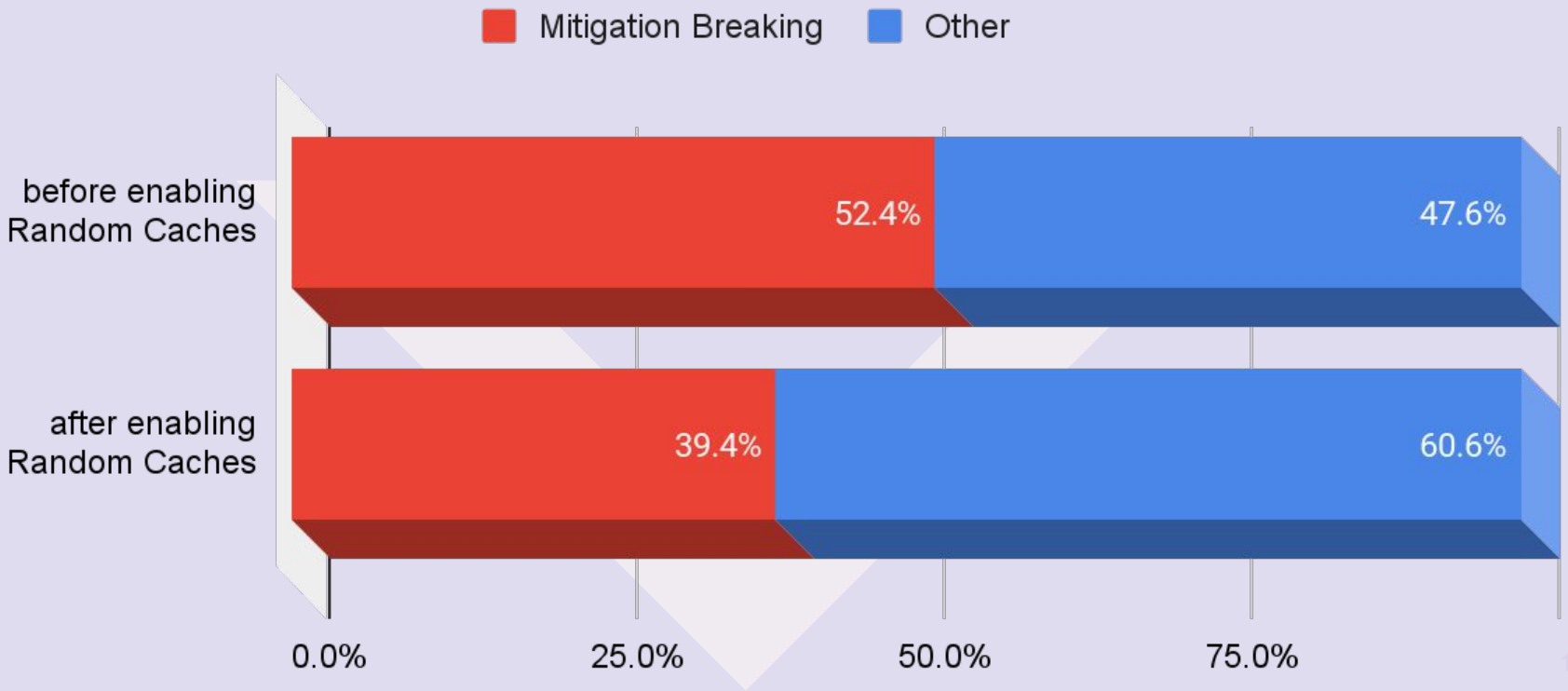
```
ffffffff813715f0 <kvmalloc_node>:
< .. snip .. >
ffffffff8137160e:       e9 dd cc 00 00          jmp    ffffffff8137e2f0 <__kmalloc_node>
ffffffff81371613:       89 f0                   mov    %esi,%eax
ffffffff81371615:       48 c1 e0 20             shl    $0x20,%rax
ffffffff81371619:       48 81 e6 ff ff ff ff    and    $0xffffffffffffffff,%rsi
ffffffff81371620:       48 09 f0                or     %rsi,%rax
ffffffff81371623:       50                      push   %rax
ffffffff81371624:       81 e6 ff ff 7f ff       and    $0xff7fffff,%esi
ffffffff8137162a:       e8 c1 cc 00 00          call   ffffffff8137e2f0 <__kmalloc_node>
< .. snip .. >
```

```
4128  void *__kmalloc_node_noprof(size_t size, gfp_t flags, int node)
4129  {
4130          return __do_kmalloc_node(size, flags, node, _RET_IP_);
4131  }
```

Mitigation Value

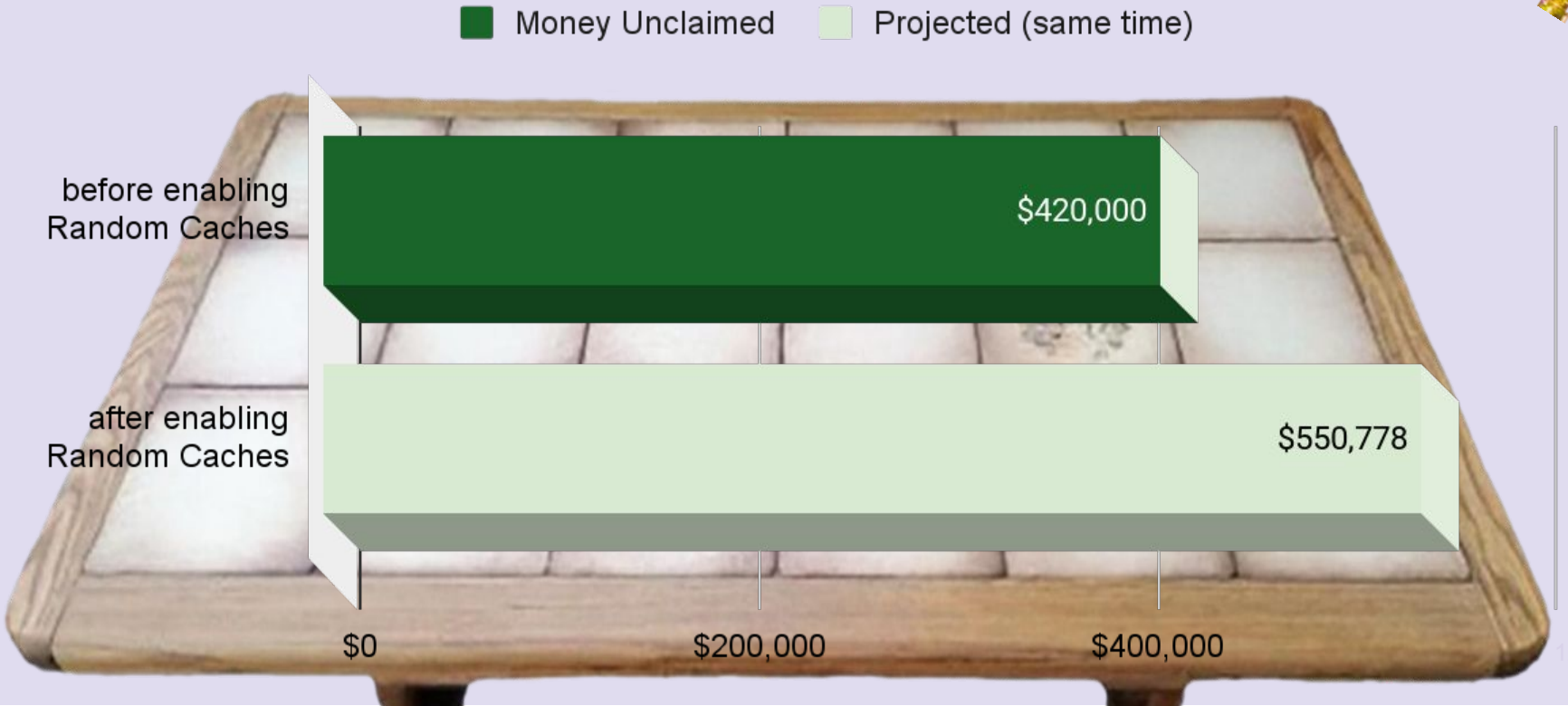
Mitigation Value

% Exploited CVEs

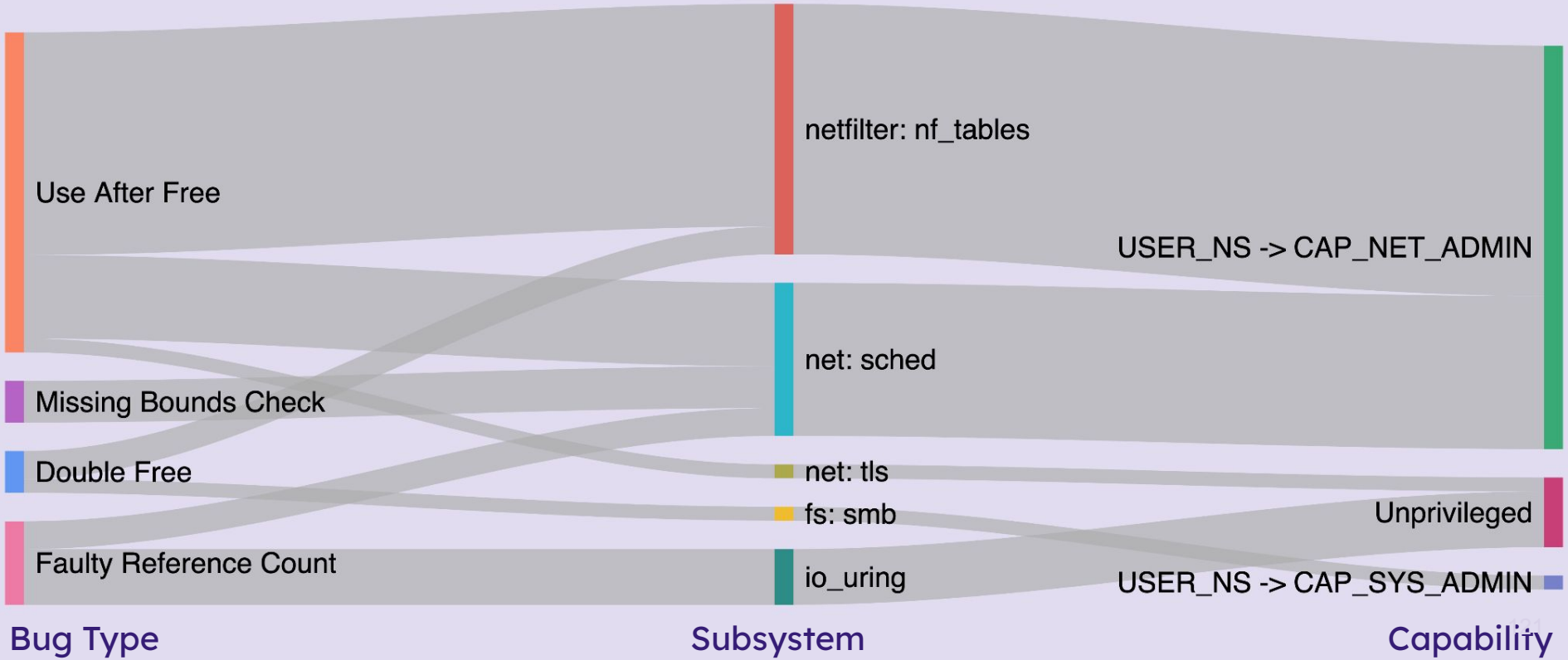


Mitigation Value

$ Unclaimed (21k per CVE)



More 🧨 for Your Mitigation



Option 1: Remove Classes of Vulnerabilities

In "Using Rust in the binder driver" at LPC23

*"It prevents mistakes with **ref counting**, **locking**, **bounds checking**, and also does a lot to **reduce the complexity** of error handling."*

Option 1: Remove Classes of Vulnerabilities

Commit Message

Alice Ryhl Nov. 1, 2023, 6:01 p.m. UTC

```
  The ultimate goal of this project is to replace the C implementation.

Signed-off-by: Alice Ryhl <aliceryhl@google.com>
---
 drivers/android/Kconfig        |   36 -
 drivers/android/binder.c       | 6630 ----------------------------------------
 drivers/android/binder_alloc.c | 1284 --------
 drivers/android/binderfs.c     |  827 -----
 4 files changed, 8777 deletions(-)
```

Option 2: Removal Attack Surface

In "io_uring: add a sysctl to disable io_uring system-wide" on LKML

*"Prevents all processes from creating new io_uring instances. **Enabling this shrinks the kernel's attack surface**."*

Option 2: Removal Attack Surface

```
4098  + static inline bool io_uring_allowed(void)
4099  + {
4100  +         int disabled = READ_ONCE(sysctl_io_uring_disabled);
4101  +         kgid_t io_uring_group;
4102  +
4103  +         if (disabled == 2)
4104  +                 return false;
4105  +
4106  +         if (disabled == 0 || capable(CAP_SYS_ADMIN))
4107  +                 return true;
4108  +
```

Option 2: Removal Attack Surface

```
4098  + static inline bool io_uring_allowed(void)
4099  + {
4100  +         int disabled = READ_ONCE(sysctl_io_uring_disabled);
4101
4102
4103
4104
4105
4106
4107
```

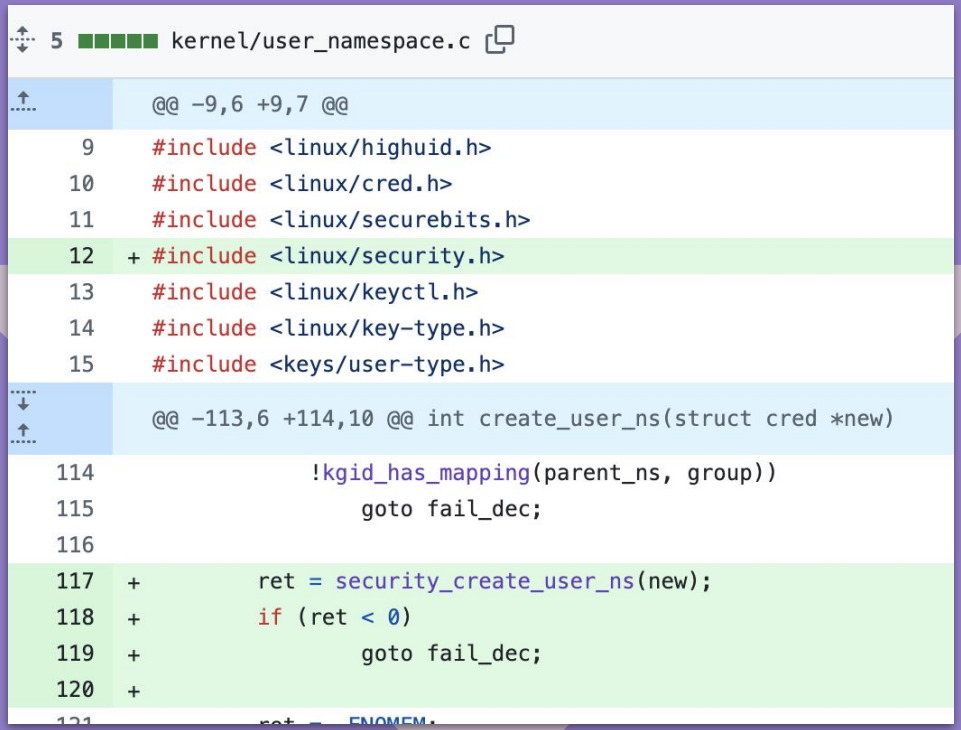
```
4116      SYSCALL_DEFINE2(io_uring_setup, u32, entries,
4117                      struct io_uring_params __user *, params)
4118      {
4119  +           if (!io_uring_allowed())
4120  +                   return -EPERM;
4121  +
4122              return io_uring_setup(entries, params);
4123      }
```

Option 3: Restrict Access to Attack Surface

In "Re: [GIT PULL] LSM patches for v6.1" on LKML

*"I personally think [letting users create user namespaces] was a mistake. We're stuck with it, but we most definitely **need knobs to manage it that isn't just [...] the kernel config**."*

Option 3: Restrict Access to Attack Surface



5 ■■■■■ kernel/user_namespace.c

```
@@ -9,6 +9,7 @@
 9   #include <linux/highuid.h>
10   #include <linux/cred.h>
11   #include <linux/securebits.h>
12 + #include <linux/security.h>
13   #include <linux/keyctl.h>
14   #include <linux/key-type.h>
15   #include <keys/user-type.h>
@@ -113,6 +114,10 @@ int create_user_ns(struct cred *new)
114                 !kgid_has_mapping(parent_ns, group))
115                     goto fail_dec;
116
117 +       ret = security_create_user_ns(new);
118 +       if (ret < 0)
119 +               goto fail_dec;
120 +
```

Option 3: Restrict Access to Attack Surface

In "A capability set for user namespaces" on LWN

*"In a world where the kernel was bug-free, user namespaces would not be a problem; in the world we actually inhabit, **they continue to worry security-oriented developers**."*

Option 3: Restrict Access to Attack Surface

```
# capsh --secbits=$((1 << 8)) --drop=cap_sys_rawio -- \
        -c 'unshare -r grep Cap /proc/self/status'
CapInh: 0000000000000000
CapPrm: 000001fffffdffff
CapEff: 000001fffffdffff
CapBnd: 000001fffffdffff
CapAmb: 0000000000000000
CapUNs: 000001fffffdffff
```

Option 3: Restrict Access to Attack Surface

```
# capsh --secbits=$((1 << 8)) --drop=cap_sys_rawio -- \
        -c 'unshare -r grep Cap /proc/self/status'
CapInh: 0000000000000000
CapPrm: 000001fffffdffff
CapEff: 000001fffffdffff
CapBnd: 000001fffffdffff
CapAmb: 0000000000000000
CapUNs: 000001fffffdffff
```

```
 static void set_cred_user_ns(struct cred *cred, struct user_namespace *user_ns)
 {
-       /* Start with the same capabilities as init but useless for doing
-        * anything as the capabilities are bound to the new user namespace.
-        */
-       cred->securebits = SECUREBITS_DEFAULT;
+       /* Start with the capabilities defined in the userns set. */
+       cred->cap_bset = cred->cap_userns;
+       cred->cap_permitted = cred->cap_userns;
+       cred->cap_effective = cred->cap_userns;
        cred->cap_inheritable = CAP_EMPTY_SET;
-       cred->cap_permitted = CAP_FULL_SET;
-       cred->cap_effective = CAP_FULL_SET;
        cred->cap_ambient = CAP_EMPTY_SET;
-       cred->cap_bset = CAP_FULL_SET;
+       cred->securebits = SECUREBITS_DEFAULT;
```

Option 3: Restrict Access to Attack Surface

```
# capsh --secbits=$((1 << 8)) --drop=cap_sys_rawio -- \
        -c 'unshare -r grep Cap /proc/self/status'
CapInh: 0000000000000000
CapPrm: 000001fffffdffff
CapEff: 000001fffffdffff
CapBnd: 000001fffffdffff
CapAmb: 0000000000000000
CapUNs: 000001fffffdffff
```

```
 static void set_cred_user_ns(struct cred *cred, struct user_namespace *user_ns)
 {
-       /* Start with the same capabilities as init but useless for doing
-        * anything as the capabilities are bound to the new user namespace.
-        */
-       cred->securebits = SECUREBITS_DEFAULT;
+       /* Start with the capabilities defined in the userns set. */
                                                ->cap_userns;
                                                cred->cap_userns;
                                                cred->cap_userns;
                                                = CAP_EMPTY_SET;
                                                CAP_FULL_SET;
                                                CAP_FULL_SET;
                                               AP_EMPTY_SET;
                                               FULL_SET;
                                               CUREBITS_DEFAULT;
```

```
+++ b/kernel/cred.c
@@ -56,6 +56,9 @@ struct cred init_cred = {
        .cap_permitted          = CAP_FULL_SET,
        .cap_effective          = CAP_FULL_SET,
        .cap_bset               = CAP_FULL_SET,
+#ifdef CONFIG_USER_NS
+       .cap_userns             = CAP_FULL_SET,
+#endif
        .user                   = INIT_USER
```

Option 3: Restrict Access to Attack Surface

```
# capsh --secbits=$((1 << 8)) --drop=cap_sys_rawio -- \
        -c 'unshare -r grep Cap /proc/self/status'
CapInh: 0000000000000000
CapPrm: 000001ffffffffff
CapEff:
CapBnd:
CapAmb:
CapUNs:
```

```
static void set_cred_user_ns(struct cred *cred, struct user_namespace *user_ns)
```

```
@@ -46,6 +103,12 @@ static void set_cred_user_ns(struct cred *cred, struct user_namespace *user_ns)
        /* Limit userns capabilities to our parent's bounding set. */
        if (iscredsecure(cred, SECURE_USERNS_STRICT_CAPS))
                cred->cap_userns = cap_intersect(cred->cap_userns, cred->cap_bset);
+#ifdef CONFIG_SYSCTL
+       /* Mask off userns capabilities that are not permitted by the system-wide mask. */
+       spin_lock(&cap_userns_lock);
+       cred->cap_userns = cap_intersect(cred->cap_userns, cap_userns_mask);
+       spin_unlock(&cap_userns_lock);
+#endif
```

```
+++ b/
@@ -56,
        .cap_effective          = CAP_FULL_SET,
        .cap_bset               = CAP_FULL_SET,
+#ifdef CONFIG_USER_NS
+       .cap_userns             = CAP_FULL_SET,
+#endif
        .user                   = INIT_USER
```

```
CAP_FULL_SET;
CAP_FULL_SET;
CAP_EMPTY_SET;
_FULL_SET;
CUREBITS_DEFAULT;
```

Mitigation Costs

Mitigation Cost - Runtime

> > What's the cost?
>
> The only thing that I see is 1-2% on kernel compilations (and "more on
> machines with lots of cores")?

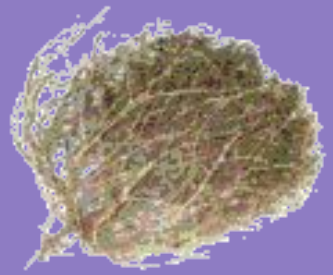
I used kernel compilation time (wall clock time) as a benchmark while preparing the series. Lower is better.

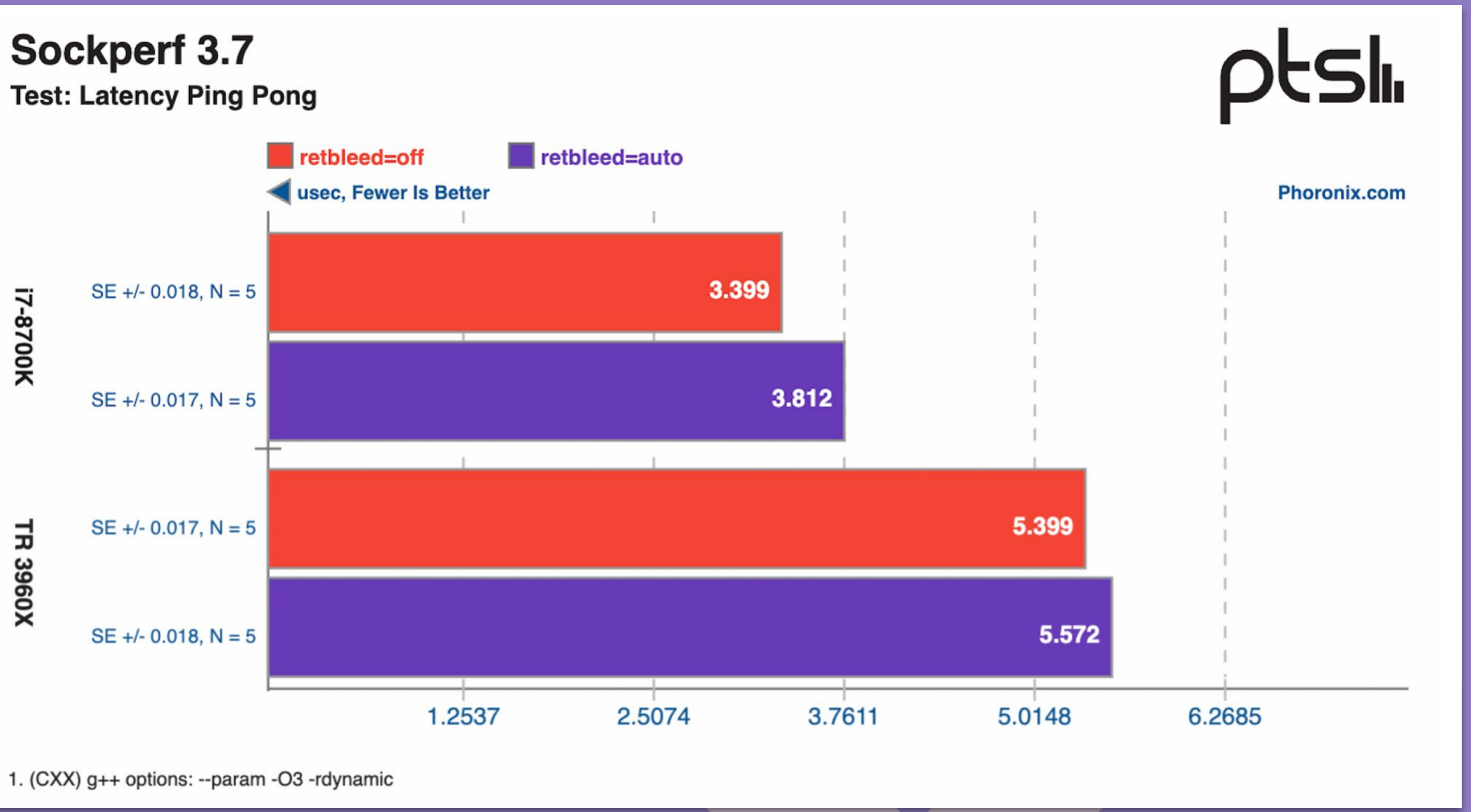
Are there any specific benchmarks that you would be interested in seeing or that are usually used for SLUB?

Mitigation Cost - Runtime

```
> > What's the cost?
>
> The only thing that I see is 1-2% on kernel compilations (and "more on
> machines with lots of cores")?

I used kernel compilation time (wall clock time) as a benchmark while
preparing the series. Lower is better.
```

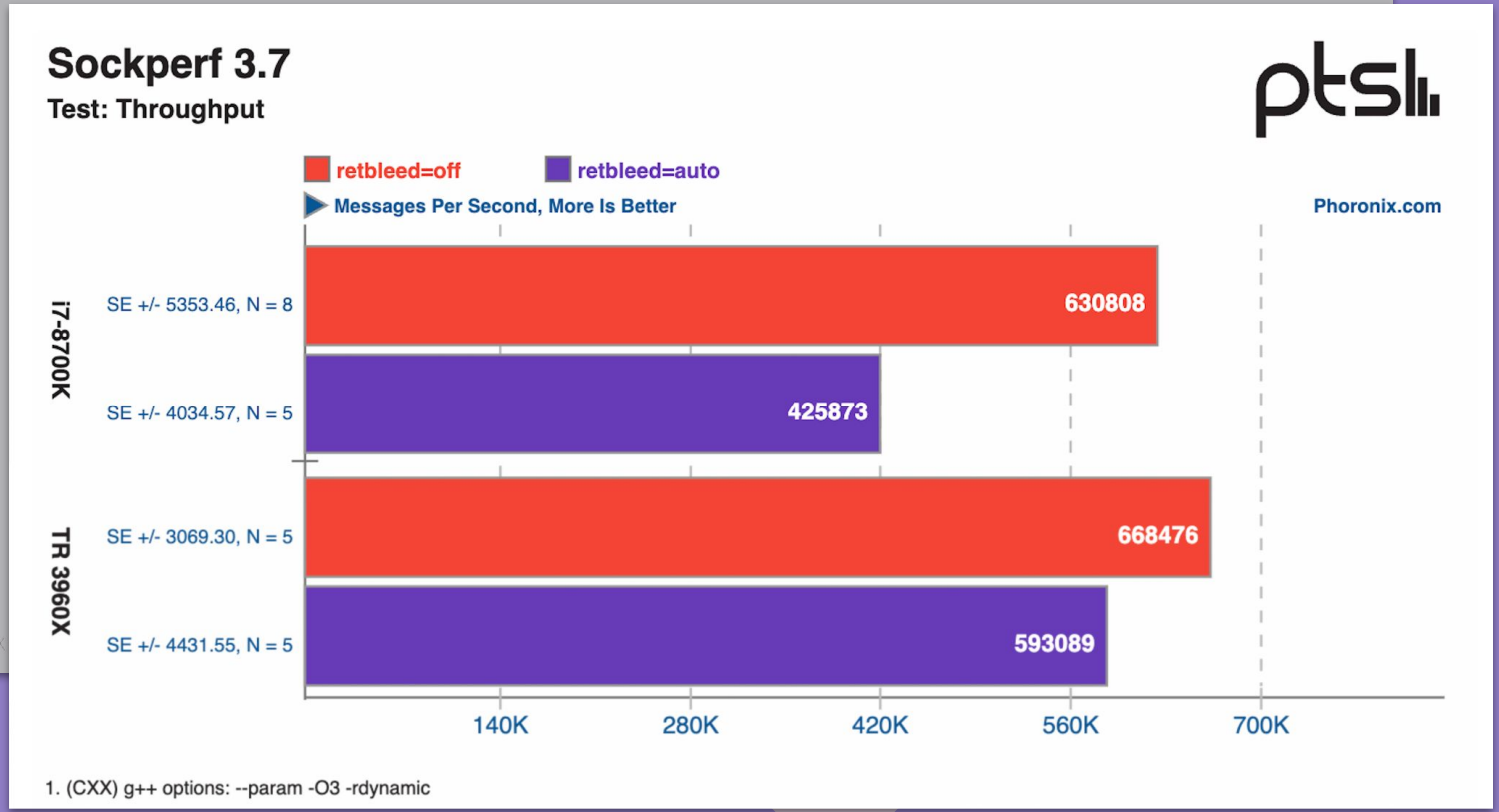
```
Are there any specific benchmarks that you would be interested in seeing or
that are usually used for SLUB?
```




Sockperf 3.7
Test: Latency Ping Pong
retbleed=off
retbleed=auto
usec, Fewer Is Better
Phoronix.com
i7-8700K
SE +/- 0.018, N = 5
3.399
SE +/- 0.017, N = 5
3.812
TR 3960X
SE +/- 0.017, N = 5
5.399
SE +/- 0.018, N = 5
5.572
1.2537
2.5074
3.7611
5.0148
6.2685
1. (CXX) g++ options: --param -O3 -rdynamic

Sockperf 3.7

Test: Latency Ping Pong



Sockperf 3.7

Test: Latency Ping Pong

Sockperf 3.7

Test: Throughput

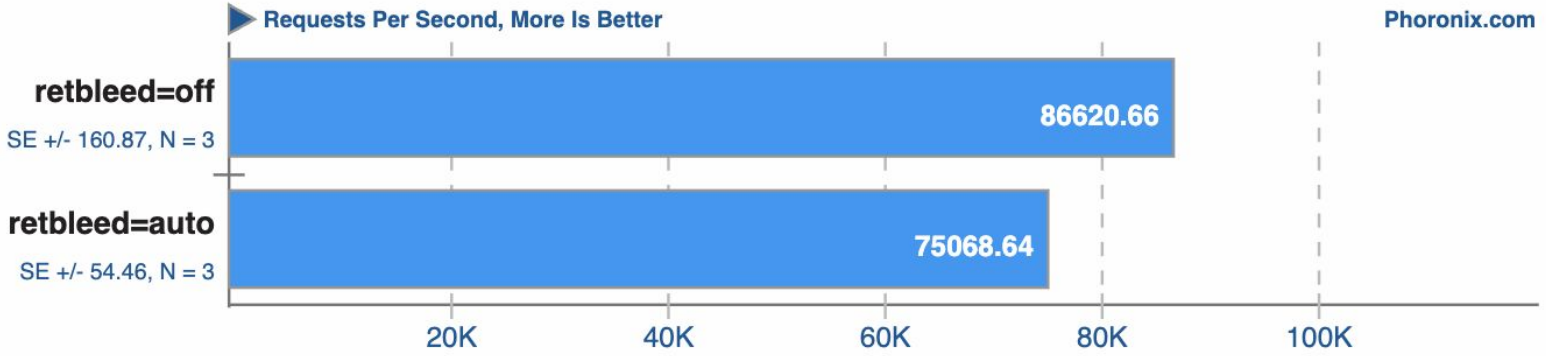
i7-8700K

TR 3960X

1. (CXX) g++ options: -param -O3 -rdynamic

nginx 1.21.1

Concurrent Requests: 500



1. (CC) gcc options: -lcrypt -lz -O3 -march=native

Sockperf 3.7

Test: Latency Ping Pong

Sockperf 3.7

Test: Throughput

nginx 1.21.1

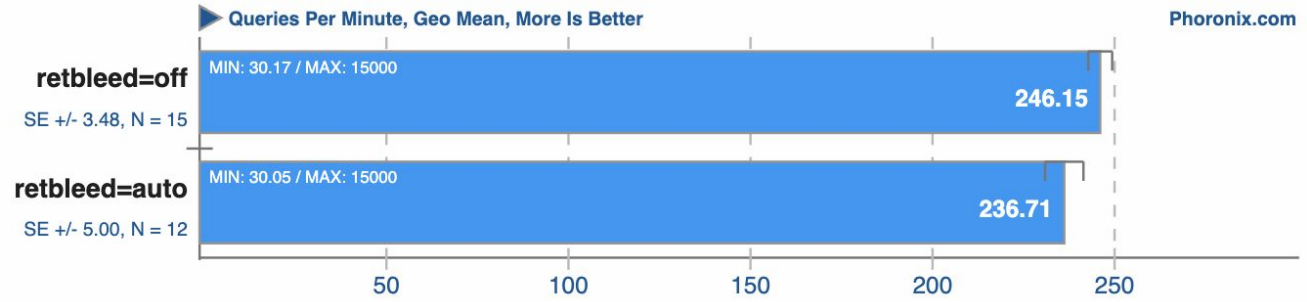
Concurrent Requests: 500

i7-8700K

TR 3960X

ClickHouse

100M Rows Web Analytics Dataset, First Run / Cold Cache



1. ClickHouse server version 22.7.1.687 (official build).

Mitigation Cost - Development

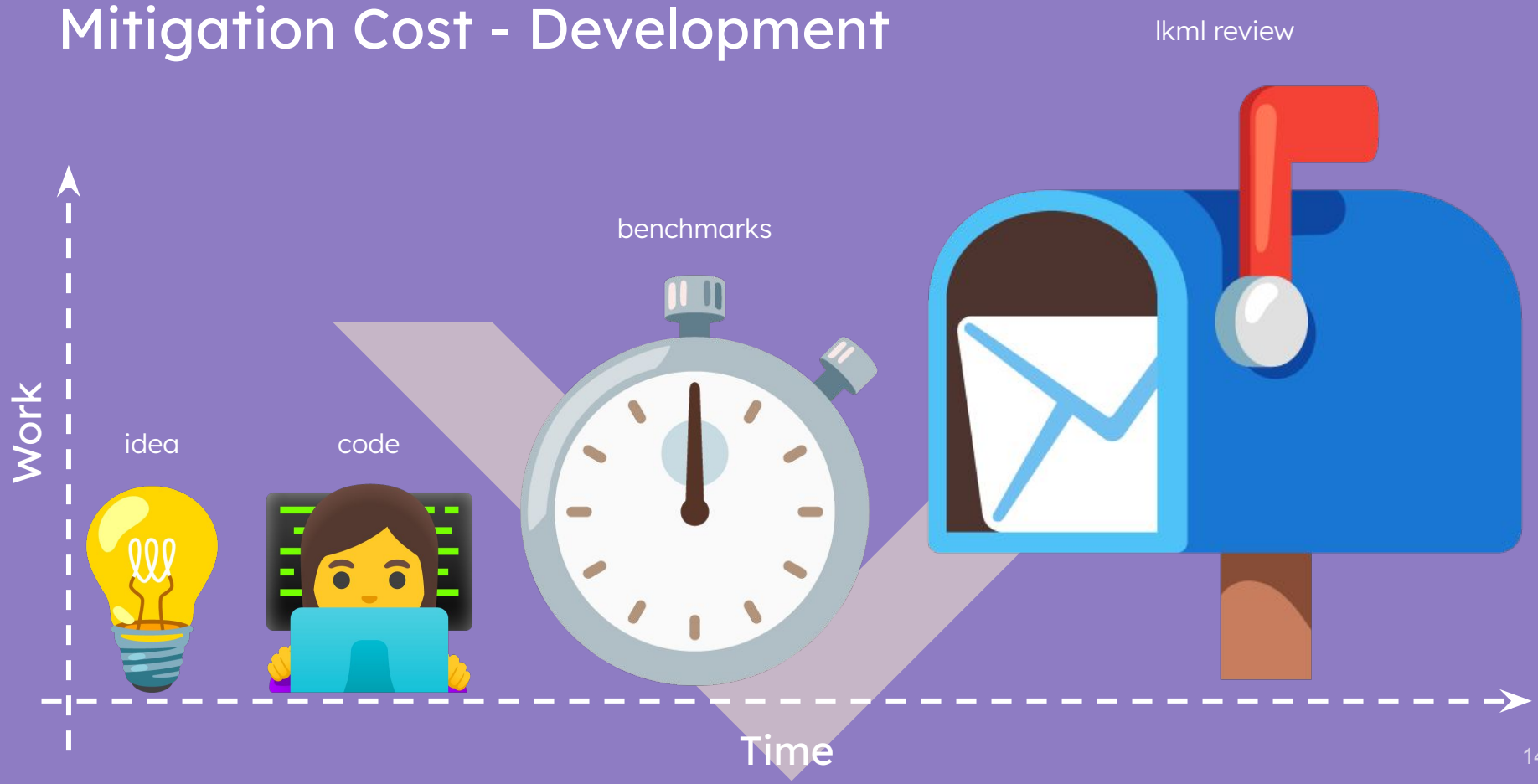
idea

code

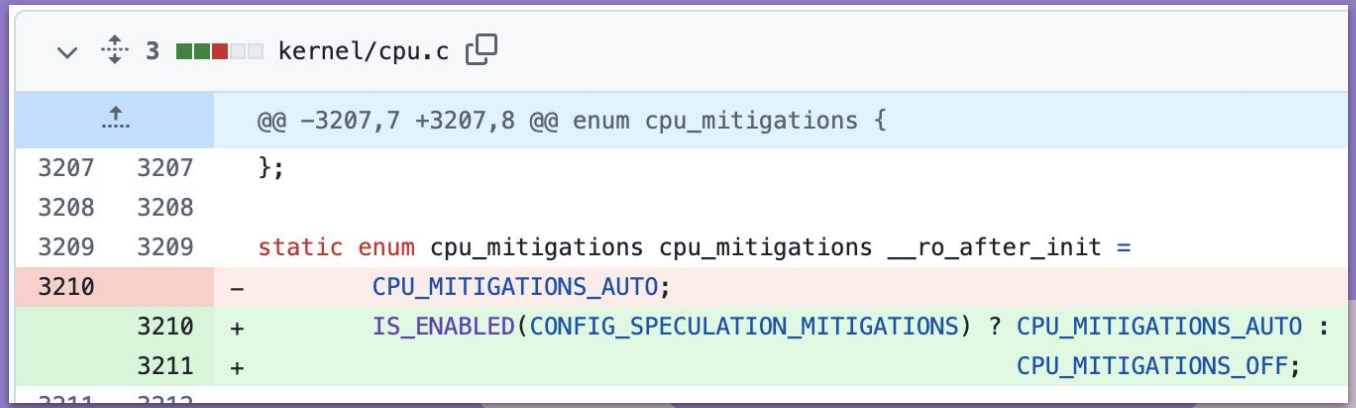
benchmarks

lkml review

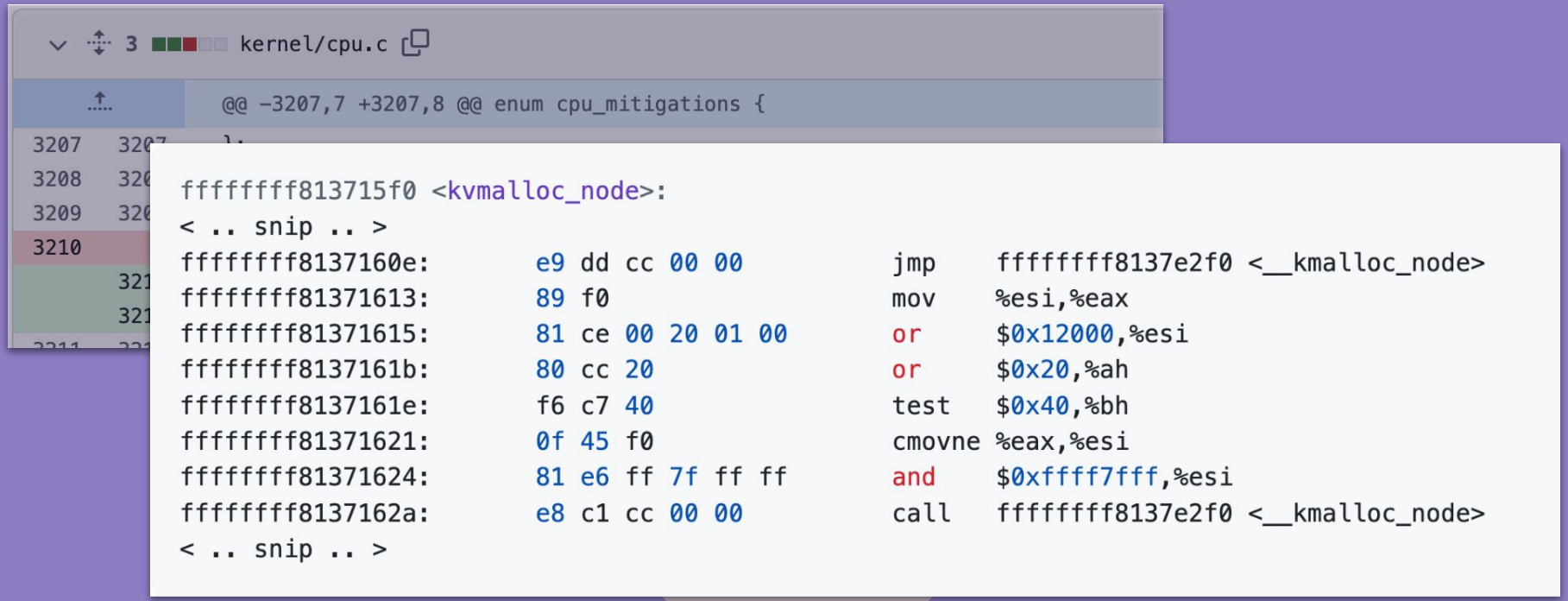
Mitigation Cost - Development



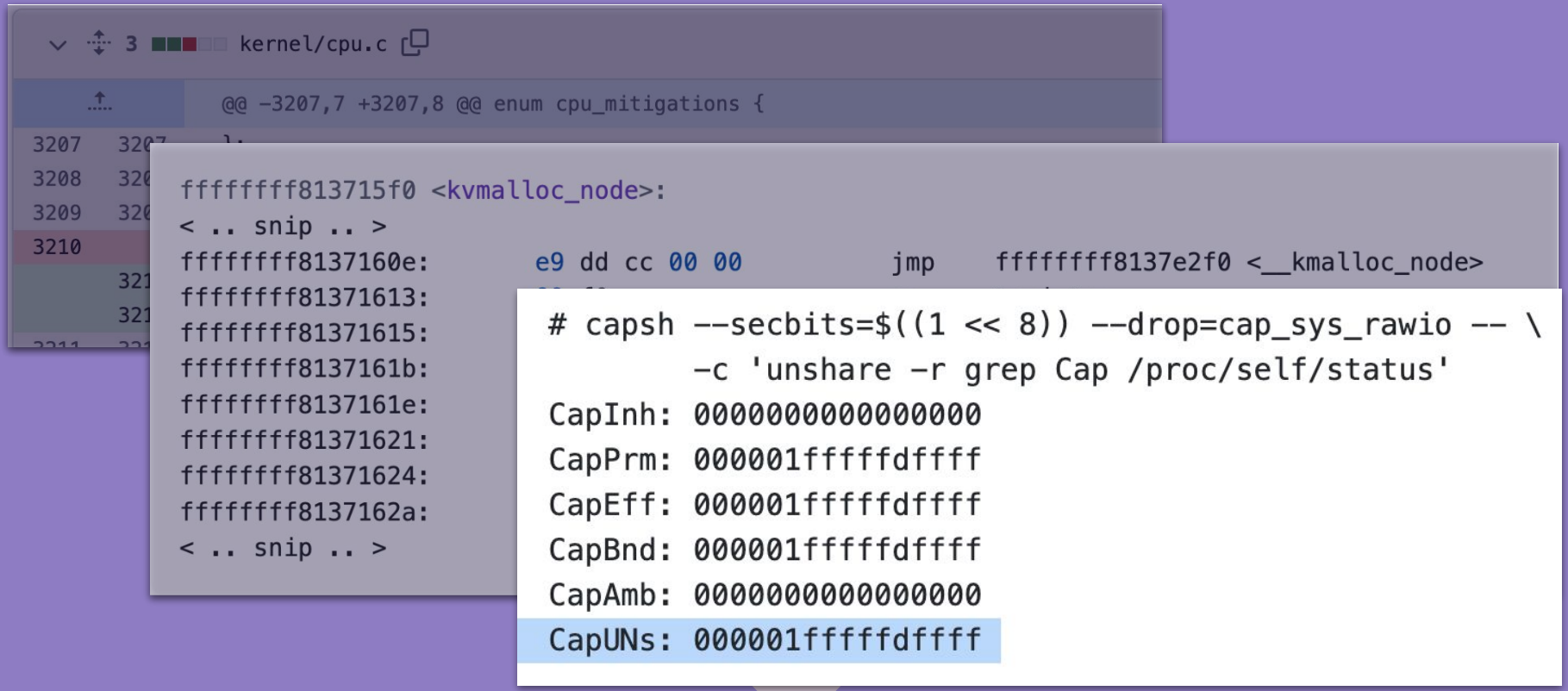
Mitigation Cost - Maintenance



Mitigation Cost - Maintenance



Mitigation Cost - Maintenance



Send Mitigations!

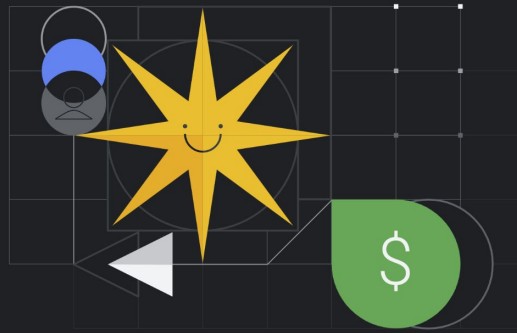
1 OUR MISSION <!--OSS Subsidies---------->

Security Subsidies

In contrast to Patch Rewards, which reward proactive security improvements after the work has been completed, Open Source Security Subsidies offer upfront financial support to provide an additional resource for open source developers to prioritize security work.

For example, if you are a small open source project and you want to improve security, but don't have the necessary resources, an Open Source Security Subsidy can help you acquire additional development capacity.

For more details on in-scope projects and our expectations, see the information on this page and the program rules.



https://bughunters.google.com/open-source-security/security-grants

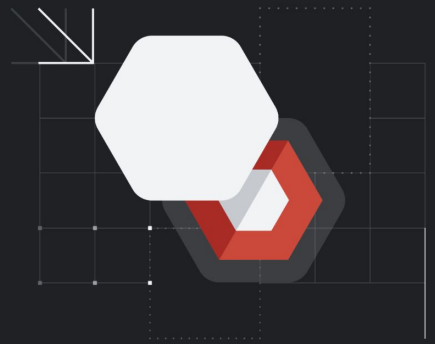
1 OUR MISSION <!--OSS Patch Rewards---------->

Patch Rewards

Through the Patch Rewards program, you can claim rewards for proactive improvements you've made to security in open source projects.

Any patch (typically a merged GitHub pull request) that you can demonstrate to have improved the security of an in-scope project will be considered for a reward.

For more details on in-scope projects and qualifying submissions, see the information on this page and the program rules.



https://bughunters.google.com/open-source-security/patch-rewards

Summary?

- **Lots of kernel vulns**
  - Tons of CVEs
  - Hard to keep up-to-date
  - Open-sourced LPE exploit for every kernel version in kCTF/kernelCTF
- **Making exploitation harder is expensive**
  - Mitigations are not perfect
  - Mitigations are expensive
- **We are trying to help find the right balance**
  - We need your help
    - Write Exploits
    - Write Mitigations

The END

sorry about the 148 slides